Adaptive space-time domain décomposition methods for Euler and Navier-Stockes equations

Oana Alexandra Ciobanu

UNIVERSITÉ PARIS 13

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS 13

Discipline: **Mathématiques Appliquées**

Laboratoire d'accueil: ONERA - Le centre français de recherche aérospatiale

Présentée et soutenue publiquement le 19 décembre 2014
par

Oana Alexandra CIOBANU

Titre

Méthode de décomposition de domaine avec adaptation de maillage en espace-temps pour les équations d'Euler et de Navier–Stokes

devant le jury composé de:

François	Dubois	Rapporteur
Laurence	Halpern	Directrice de thèse
Raphaèle	Herbin	Rapporteure
Xavier	Juvigny	Examinateur
Olivier	Lafitte	Examinateur
Juliette	Ryan	Encadrante

UNIVERSITY PARIS 13

THESIS

Presented for the degree of

DOCTEUR DE L'UNIVERSITÉ PARIS 13

In **Applied Mathematics**

Hosting laboratory: ONERA - The French Aerospace Lab

presented for public discussion on 19 decembre 2014
by

Oana Alexandra CIOBANU

Subject

Adaptive Space-Time Domain Decomposition Methods for Euler and Navier–Stokes Equations

Jury:

François	Dubois	Reviewer
Laurence	Halpern	Supervisor
Raphaèle	Herbin	Reviewer
Xavier	Juvigny	Examiner
Olivier	Lafitte	Examiner
Juliette	Ryan	Supervisor

newpage

Remerciements

Tout d'abord, je remercie grandement Juliette Ryan pour avoir accepté d'être mon encadrante de stage puis mon encadrante de thèse. Pendant plus de trois ans, elle m'a fait découvrir mon métier de jeune chercheuse avec beaucoup de patience et de professionnalisme. Elle m'a soutenue, elle a été d'une disponibilité et d'une écoute extraordinaires, tout en sachant être rigoureuse et exigeante avec moi comme avec elle-même. Humainement, j'ai beaucoup apprécié la relation d'amitié que nous avons entretenue, le climat de confiance que nous avons maintenu et les discussions extra-mathématiques que nous avons pu avoir et qui ont renforcé le lien que nous avions. Je la remercie pour tout ça, et je sais que j'en oublie.

Je remercie sincèrement Raphaèle Herbin et François Dubois d'avoir accepté la lourde tâche d'être mes rapporteurs, et je les remercie pour l'attention qu'ils ont porté à mon travail dans un délai plus que raisonnable.

C'est un plaisir et un honneur pour moi d'avoir dans mon jury Olivier Lafitte. C'est grâce à lui que je suis arrivée en MACS, et je suis très fière de le retrouver pour cette soutenance de thèse, c'est une jolie manière de boucler la boucle.

Je remercie également mes collègues de l'Onera, pour les nombreuses discussions que nous avons eues au cours de ces trois années et pour les moments sympathiques que nous avons passés au laboratoire. En particulier je voudrais remercier Xavier Juvigny, tout d'abord parce qu'il a suivi de très près l'avancée de mon travail, et surtout pour sa disponibilité et ses facultés pour me dépatouiller de tous mes problèmes informatiques. J'ai eu énormément de plaisir à le côtoyer durant ces trois années, à échanger avec lui sur la thèse, sur la vie, et sur plein de choses en général. Sans lui, la thèse n'aurait pas été drôle du tout et je suis vraiment contente d'avoir fait sa connaissance et de m'être lié d'amitié avec lui.

Je vais à présent remercier mes proches. Je remercie mes parents et mes frères pour leur soutien et la confiance aveugle qu'ils ont eue envers ma réussite. Je remercie Mikael, Roxana, Raluca, Luciana, Diana et Alina pour leur soutien moral. Je remercie Roxana pour avoir relu une partie de ma thèse et pour leurs corrections, Alina pour l'aide qu'elle m'a apportée pour préparer le pot, et Mikael pour les deux.

Je tiens à exprimer ma profonde reconnaissance à Laurence Halpern, qui m'a fait l'honneur d'être ma directrice de thèse et pour toutes les discussions constructives et conseils autour mon travail.

Ma thèse ne se serait sans doute jamais achevée sans leur soutien et je les remercie pour tous les moments qu'on a passés ensemble.

Contents

Introduction

This work focuses on the research field of laminar flow of an ideal gas around solid bodies, on the resolution of aerodynamic multi-scale problems that are costly and difficult to solve in their original form. In the aerodynamic field, scientists are facing many different problems, geometrical problems due to the complex design of civil aircraft, military aircraft, helicopters, space transport, missile systems, launchers, air intakes, nozzles, propulsive jets and other bodies in movement. Other problems are issued from the model used to describe the physical configuration. Most of the models are robust only for particular cases, coefficients are established on particular cases as well, for simplified models and small range data problems. Thus, phenomena too complex to be reliably predicted by theory and too dangerous or expensive to be reproduced in the laboratories must be simulated by computational scientists. A simulation with low range precisions can lead to false results, unrealistic behaviour and even damages if used in experimental work. At all levels, in order to better approach the geometry and to better validate a model, for physical viability precision, large data systems must be solved. Several techniques of parallel computing have been developed as they allow to solve large data systems of equations. They essentially depend on the chosen domain decomposition method, but convergence problems may occur for large numbers of sub-domains.

In what concerns the numerical point of view on solving the Navier–Stokes system of equations, robust and fast methods are now available, which combine non-linear and linear solvers requiring less memory capacity. In the context of long term simulations, the class of global implicit approaches has proved its superiority as they are able to simulate a quasi-steady-state behaviour without being restricted to short time steps to ensure convergence. However, in industrial applications explicit methods are preferred, as they are easy to compute and to adapt, but very expensive since they are restricted to a small time step. Between the range of existing algorithms the choice of an appropriate one is still uncertain. The performance of both, explicit and implicit methods can be improved when combined with a domain decomposition method.

Domain decomposition methods split large problems into smaller sub-problems that can be solved in parallel. Space domain decomposition method is used to provide high-performing algorithms in many fields of numerical applications. Yet, the technology evolution towards multi-cores, multi-processors, GPU and multi-GPUs is not completely exploited. To achieve full performance on large clusters with up to 100 000 nodes (such as recently the IBM Sequoia, or GPUs) the time dimension has to be taken into account. An essential gain to be obtained from time-space domain decomposition is the ability to apply different time-space discretisations on sub-domains thus improving efficiency and convergence of implicit schemes. The idea of domain decomposition is quite old, it was proposed for the first time in 1869 by H.A. Schwarz [87] with the purpose of solving a linear problem (Laplace equation) over a geometry, which it was impossible to solve in its initial form. Schwarz splits the global problem into two overlapping problems and solves them alternately by exchanging interface conditions. He manages in this way to solve for the first time the Laplace equation for a complex geometry and gives the main goal to the domain decomposition method, the one of managing complicated geometries. More than one century later, in a series of papers [64, 65, 66], P.-L. Lions revitalises Schwarz idea and proposes a parallel formulation of what is known today as Parallel Schwarz. He discusses

overlapping and non-overlapping coincident domain decomposition methods and different types of interface conditions, and this for linear systems of equations. Lions manages to completely parallelise sequential problems, but some problems still diverge for large number of sub-domains. Another technique developed in the same period by J.L. Steger and J.A. Benek [91, 90] is the chimera overset grid methods. It is an additive Schwarz method (as shown in the paper of Brezzi and co [10]) for overlapping domains to adapt body conforming structured grid methods to complex geometries. It allows moving one body with respect to another or adding new bodies into the domain without mesh reconstruction, by allowing meshes to move along with one body. The connection between grids is usually made by interpolation.

Our attention focuses on the improvement of the Schwarz Waveform Relaxation (SWR) method, introduced under this name by Gander [38] at the 10th Domain Decomposition Conference to solve parabolic equations. The Waveform Relaxation method (Lelarasmee and al. [62]) is an iterative method to solve time dependent problems. Each iteration produces a better approximation of the solution over the whole time interval. This scheme was extended to a multi-splitting formulation on overlapping sub-domains by Bjorhus [5], Jeltsch and Pohl [58] and Gander and Stuart [46]. Originally applied to linear PDEs, the SWR algorithm was extended and optimised for the non-linear reactive transport equations by Haeberlein [49, 50] and for the incompressible Navier–Stokes equations by Audusse and al [2]. With the SWR method different time-space discretisations can be applied on sub-domains thus improving efficiency and convergence of the schemes.

Schwarz based domain decomposition methods are proved to be efficient mostly for linear problems. Yet, some convergence problems still appear for large number of sub-domains (for example in Finite Element methods with over 100 sub-domains). For large number of sub-domains of small size, some parallel simulations with elsA, the Onera's software for aerodynamic, lead to loss of solution accuracy. In this work we conduct a numerical study of Schwarz based domain decomposition methods and especially the SWR method for the Euler and Navier–Stokes system of equations. We aim to identify the loss of accuracy problem encountered inside Onera and help eliminate their drawbacks. We propose an improved parallel time-space method for steady/unsteady problems modelled by Euler and Navier-Stokes equations for a direct numerical simulation that uses the SWR method. Within the SWR iterative process, we propose an adaptive time stepping technique to improve the scheme consistency. By adaptivity we mean that, for each SWR iteration, as we improve the coupling conditions we adapt the time step to satisfy the CFL condition and ensure stability of our method, thus different time steps in each sub-domain and inside each time window.

The main aim of this study is to build a parallel in space and in time CFD Finite Volumes code for steady/unsteady problems modelled by Euler and Navier-Stokes equations based on Schwarz/chimera method that improves consistency, accelerates convergence and decreases computational cost. Another aim is to build this improvement in a modular form so as to be easily integrated in elsA or any other platform.

This work is structured in four chapters as presented below.

Organisation of the Manuscript

Chapter 1

The first chapter is dedicated to the framework of this study. It begins with the presentation of the Euler and Navier–Stokes equations, the solving of which will be the application of our problem. We present and discuss the model, the time and space discretisation and the physical boundary treatment. A bibliographical study is made to help choosing the most adapted numerical scheme to compute the numerical fluxes. Once the space discretisation is fixed we continue with the presentation of the time discretisation via two different classes of methods:

Chapter 2

In the second part of this work, we study numerical techniques and algorithms that apply to parallelism on different possible architectures. We present the context of parallel computing through a global and summarised review of different domain decomposition approaches. We focus on domain decomposition methods based on Schwarz methods. We give formulations of classical domain decomposition methods (alternating Schwarz method, parallel Schwarz method, Schwarz Waveform Relaxation method) applied to the Navier–Stokes system of equations for both explicit and implicit class of solvers. We then formulate the problem we are aiming to solve: improvement of a domain decomposition method. We propose a new flexibility to the SWR method aiming to improve the scheme consistency and accelerate its convergence.

In the last part of this chapter we discuss the different types of transmission conditions on artificial boundaries issued by the domain decomposition. The implementation of a Robin type boundary condition and values of different parameters are discussed.

Chapter 3

In the third chapter we discuss the means of parallel computing, how to understand, organize and tune the parallel methods. We talk about evolution of computer architecture (shared memory, distributed memory and hybrid), how to exploit its capacities and make sure that the algorithms are flexible and portable. When different architectures are possible, a developer must reevaluate, rethink the problem and be able to appreciate its adaptivity to these architectures. We briefly present the tools (OpenMP, MPI and GPU CUDA) that enable the parallel computation, their evolution, advantages and limits. Example of applications are shown for different computer architectures. In the last section we discuss ways of measuring parallel efficiency.

Chapter 4

The last chapter of this manuscript is dedicated to the validation of the presented methods. Validations and comparisons of different numerical schemes to compute the Euler fluxes are conducted. Then, we validate the domain decomposition methods applied to explicit and implicit methods. A numerical convergence study and comparisons of different Schwarz based decomposition methods is done over three main cases:

- The 2D isentropic vortex evolution based on Yee's paper [99]. The problem of numerical diffusion of vortices can appear when dealing with simulations of aircraft trailing vortices, blade–vortex interaction of helicopter rotors, rotor–stator interaction of turbo-shaft engines, aeroacoustic problems or weather forecasting. Moreover, this test is an exact solution of the Euler equations and provides an accurate study of convergence.
- More often studied in the aeroacoustic fields, the sound generation in a 2D low Reynolds mixing layer, based on [24, 55], is a very unsteady case that requires small time steps simulations;
- The vortex shedding around rectangles [28, 80, 92, 9, 86] has applications such as the aerodynamic drag reduction for air-planes, road vehicle, damage predictions for inclined air-foils, ocean pipe line or risers, off- shore platform supports, suspension bridges, steel towers or smoke stacks, etc.

Performance of the different parallel computing strategies (using OpenMP, MPI) are compared for each test. A mono-domain application on GPU is also presented in this chapter.

Chapter 1

Numerical Methods for Euler and Navier–Stokes Equations

Contents

In 1755, Leonhard Euler gives a set of equations governing inviscid flows and, in 1822 Henri Navier, and independently George Stokes in 1845, formulates the central equations to model the fluid dynamics adding the viscosity to the Euler system of equations. In the first part of this chapter we introduce a general presentation of the fluid dynamics equations, then we set-up the numerical frame. We present separately the space discretisation and the time discretisation of the exact problem. The space discretisation is of Finite Volume type, thus it:

- allows to handle arbitrary geometries,
- is especially adaptable to Cartesian grids,
- requires no additional treatment on a composite grid,
- is also a common technique inside Onera and its main software.

The retained Finite Volume formulation for both compressible Euler and Navier–Stokes equations is based on dimensionless variables. This step allows the reduction of the number of parameters and facilitates the study of the fluid behaviour. Before discussing the numerical schemes, a theoretical introduction is given together with the definition of the Riemann problem. Next, a suitable numerical scheme for modelling the Euler fluxes has been chosen among the literature. The motivation for this section is the need for accurate simulations of flows with shocks which exist in many fields of physics. As Woodward mentions in [98] *Much experience indicates that the overall accuracy of such simulations is very closely related to the accuracy with which flow discontinuities are represented.*

The time integration is first done explicitly using a first order Euler Method and a second order Runge-Kutta method, and second in an implicit manner using a second order multi-step Backward Differentiation Method. The solving of the non-linear system of equations issued of a Backward Differentiation Method is solved with an Inexact Newton method combined with one of the three presented linear schemes. We add no new contribution to this chapter, however the presentation of the numerical schemes choices is necessary. All main definitions, notations and notions are going to be presented in this first chapter.

1.1 Fluid Dynamics Equations

We consider the time-dependent compressible Navier–Stokes equations, present the equations, different notions and notations that will be referred in the entire manuscript. Let p denote the pressure, ρ the mass density, $u,\ v,\ w$ the x, y, respectively z-component of velocity and E the total energy per unit. We note $\vec{u} = (u, v, w)^T$, the velocity vector. The basic equations of fluid dynamics can be written with the primitive variables $\rho,\ u,\ v,\ w$ and p, and for an arbitrary domain $\Omega \subset \mathbb{R}^d, d \in \{1, 2, 3\}$ and without any external forces as a set of three equations describing mass conservation, momentum conservation and energy conservation :

$$\left\{\begin{array}{ll} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) & = 0 \\ \frac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\vec{u} \otimes (\rho \vec{u})) + \nabla \cdot pI - \nabla \cdot \tau & = 0 \\ \frac{\partial \rho E}{\partial t} + \nabla \cdot (\vec{u}(\rho E + p)) - \nabla \cdot (\tau \vec{u} - q) & = 0 \end{array}\right. , \tag{1.1}$$

where τ is the deformation tensor and q the heat flux. The conservation laws imply that during the evolution of the fluid, at all times, mass, generalized momentum and energy are conserved. These three conditions completely determine the behavior of the system. We still have to add the specifications of the nature of the fluid that act on the momentum and energy equations. In order to determine all terms of these equations, the sources influencing the variation of the momentum must be defined. These forces are the external volume forces (not considered in this work) and the internal forces that depend on the nature of the fluid considered.

We assume that the fluid is Newtonian (*the time rate of change of the momentum in* Ω *is equal to the total force acting on the volume* Ω*, the time rate of change of total energy is equal*

to the work done, per unit time, by all the forces acting on the volume plus the influx of energy per unit time into the volume), then the viscous stress tensor of constraints is an isotropic linear function depending on the deformation tensor given by:

$$\tau = \lambda(\nabla \cdot \vec{u})I + 2\mu D,$$

where $D = \frac{1}{2}(\nabla\vec{u} + (\nabla\vec{u})^T)$ is the deformation tensor, λ and μ are the viscosity coefficients of the fluid that we suppose given by the Stokes hypothesis:

$$3\lambda + 2\mu = 0.$$

μ (measured in $\frac{kg}{m.s}$) is the dynamic viscosity. The total energy per unit, E is given by the sum of the internal energy per unit mass e and the specific kinetic energy per unit mass $\frac{1}{2}u^2$.

$$E = \frac{1}{2}u^2 + e.$$

The heat flux is given by the Fourier law:

$$q = K_\top \nabla \top,$$

where $K_\top = c_p\dfrac{\mu}{Pr} = \dfrac{\gamma r_{gas}}{\gamma - 1}\dfrac{\mu}{Pr}$ is the conductivity coefficient and Pr is the Prandtl number (assumed to be constant), $\top = \frac{\gamma-1}{r_{gas}}e$ is the absolute temperature, $\gamma = \frac{c_p}{c_v}$ is the specific heat ratio between c_p, the heat specific for a constant pressure and c_v the specific heat for a constant volume, r_{gas} represents the ratio of the universal constant of perfect gases to the molar mass of the gas considered. For an ideal gas with constant specific heats of ratio $\gamma = 1.4$, the universal constant of ideal gases r_{gas} equals 8.31, the molar mass of a perfect gas has the value 0.029 and the conductivity coefficient $Pr = 0.72$.

In order to avoid rounding effects, we solve a non-dimensional form of the system which is obtained by a division of the dimensional quantities by some reference factors. The reference factors are issued from the normalization of the dimensional quantities.

$$\rho \leftarrow \frac{\rho}{\rho_{ref}}, \quad \vec{u} \leftarrow \frac{\vec{u}}{\vec{u}_{ref}}, \quad p \leftarrow \frac{p}{\rho_{ref}\vec{u}^2_{ref}}, \quad E \leftarrow \frac{E}{\vec{u}^2_{ref}}, \quad \mu \leftarrow \frac{\mu}{\mu_{ref}},$$

- ρ_{ref}– fluid inflow density,
- $\vec{u}_{ref}$– fluid inflow velocity,
- μ_{ref}– fluid inflow dynamic viscosity.

From (1.1) we obtain the normalized Navier-Stokes system of equations:

$$\begin{cases} \frac{\partial\rho}{\partial t} + \nabla \cdot (\rho\vec{u}) = 0, \\ \frac{\partial\rho\vec{u}}{\partial t} + \nabla \cdot (\vec{u} \otimes (\rho\vec{u})) + \nabla \cdot pI - \frac{1}{Re}\nabla \cdot \tau = 0, \\ \frac{\partial\rho E}{\partial t} + \nabla \cdot (\vec{u}(\rho E + p)) - \frac{1}{Re}\nabla \cdot (\tau\vec{u} - q) = 0, \end{cases} \tag{1.2}$$

where $Re = \frac{\rho u L}{\mu}$ is the dimensionless Reynolds number, the ratio of inertial to viscous forces. Here, L is the characteristic length. This study is focused on laminar flows ($Re < 2300$), where no additional information is required. However, in computational fluid dynamics, most of the flow situations occurring in nature enter into a form of instability, called turbulence. This situation appears for some Reynolds numbers ($Re > 4000$) that are not going to be studied in this work. The transition between laminar and turbulent flows is made by the transient flow ($2300 < Re < 4000$).

In our study, when we refer to the Navier-Stokes system of equations we are implicitly considering the normalised Navier-Stokes system of equations.

1.1.1 Compressible Navier–Stokes Equations

In the absence of external forces, the dimensionless compressible Navier–Stokes system of equations can be written under the conservative form and for an arbitrary volume Ω:

$$U_t + F_{Euler}(U)_x + G_{Euler}(U)_y + H_{Euler}(U)_z + F_{vis}(U)_x + G_{vis}(U)_y + H_{vis}(U)_z = 0, \quad (1.3)$$

where U is the column vector containing the conservatives variables:

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}.$$

The amount of the quantity which crosses the boundary surface is called the flux and is issued from two contributions. One from the convective transport, also mentioned as an Euler flux of the fluid. The other is due to the the molecular motion, which is always present, even if the fluid is at rest and acts as a diffusive effect and it is also called the viscous flux. The convective or Euler fluxes $F_{Euler}(U)$, $G_{Euler}(U)$, $H_{Euler}(U)$ are:

$$F_{Euler}(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho u(E+p) \end{pmatrix}, G_{Euler}(U) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho v(E+p) \end{pmatrix}, H_{Euler}(U) = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho w(E+p) \end{pmatrix}.$$

The diffusive or viscous fluxes $F_{vis}(U)$, $G_{vis}(U)$, $H_{vis}(U)$ are:

$$F_{vis}(U) = \begin{pmatrix} 0 \\ \tau_{11} \\ \tau_{21} \\ \tau_{31} \\ u\tau_{11} + v\tau_{12} + w\tau_{13} - q \end{pmatrix}, G_{vis}(U) = \begin{pmatrix} 0 \\ \tau_{12} \\ \tau_{22} \\ \tau_{32} \\ u\tau_{21} + v\tau_{22} + w\tau_{23} - q \end{pmatrix},$$

$$H_{vis}(U) = \begin{pmatrix} 0 \\ \tau_{13} \\ \tau_{23} \\ \tau_{33} \\ u\tau_{31} + v\tau_{32} + w\tau_{33} - q \end{pmatrix},$$

with τ_{ij}, $1 \leq i, j \leq 3$, the coefficients of the deformation tensor.

$$\begin{cases} \tau_{11} = \mu(-\frac{2}{3}(u_x + v_y + w_z) + 2u_x), \\ \tau_{12} = \tau_{21} = \mu(u_y + v_x), \\ \tau_{13} = \tau_{31} = \mu(u_z + w_x), \\ \tau_{23} = \tau_{32} = \mu(v_z + w_y), \\ \tau_{22} = \mu(-\frac{2}{3}(u_x + v_y + w_z) + 2v_y), \\ \tau_{33} = \mu(-\frac{2}{3}(u_x + v_y + w_z) + 2w_z). \end{cases}$$

We note that, under the previous form and with the described hypothesis, the system to solve, 1.3, consists five equations associated to six unknowns (ρ, u, v, w, E, p). In the case of low temperature variations, like the different phases a flying airplane passes through, a perfect gas model can be used to complete the problem. For a perfect gas model the pressure is given by the equation of state:

$$p = (\gamma - 1)\rho(E - \frac{1}{2}u^2).$$

We can separate the convective fluxes as first order derivative terms in space that lead the transport properties of the flow and the diffusive fluxes given by second order derivative terms in space which express the molecular diffusion of the flow. The unsteady Navier–Stokes system of equations can be essentially parabolic in time and space or parabolic-hyperbolic. The steady Navier–Stokes system of equations is elliptic-hyperbolic. Global existence theorems for compressible Navier–Stokes equations exist for small initial data ([79]) and under certains constraints (see [37] and references within).

1.1.2 Euler Equations

If we neglect the viscous term in the Navier–Stokes system of equations which thus becomes the Euler equations, we get the most general flow model for a non viscous, non heat conducting fluid. The system of equations is reduced from second order to first order. This important property will determine the numerical approach. For the conservative variables U and for an arbitrary domain Ω, the time-dependent Euler equations are given, in their condensed form by:

$$U_t + F_{Euler}(U)_x + G_{Euler}(U)_y + H_{Euler}(U)_z = 0. \tag{1.4}$$

It is a system of conservation laws in differential form, a first-order partial differential equations purely hyperbolic in space and time, independently of the flow regime, subsonic or supersonic. We can describe a hyperbolic system as a system that describes convection phenomena. Another important property is that Euler equations allow discontinuous solutions in certain cases, as we will see further in the case of shock waves occurring in supersonic flows. In this case, a local form of the conservation laws over a discontinuity, called the Rankine–Hugoniot relations, needs to be respected.

In order to integrate in time, an initial and a boundary solution need to be imposed. The first step from an initial mathematical model to a final numerical solution is to select the level of approximation of the physical problem to be solved, as done previously. The second step is the choice of the discretisation method with two sub choices: *which space scheme?, which time scheme?*, selection between finite difference, finite element or finite volume methods, selection of the order of accuracy of the spatial and the time discretisation. The third step is the selection and the analysis of a resolution method of the non-linear system and for the associated linear systems. Finally, analysis of the full scheme has to be made in terms of stability and convergence properties. We will start by briefly presenting the steps to achieve the weak formulation of the problem using a finite volume technique.

1.2 Finite Volume (FV) Method

Let us note:

$$\Psi_{conv}(U) = \big(F_{Euler}(U), G_{Euler}(U), H_{Euler}(U)\big), \tag{1.5}$$

the total convective or Euler flux and

$$\Psi_{vis}(U, \nabla U) = \big(F_{vis}(U), G_{vis}(U), H_{vis}(U)\big), \tag{1.6}$$

the total viscous flux. Let us now rewrite the non-dimensional Navier–Stokes equations under the following condensed form:

$$U_t + \nabla \cdot \Psi_{conv}(U) + \nabla \cdot \Psi_{vis}(U, \nabla U) = 0, \tag{1.7}$$

on Ω, the computation domain, of boundary $\partial\Omega$, moving in an absolute reference frame. We discretise Ω into N non-overlapping cells:

$$\Omega = \bigcup_{i=1}^{N} \Omega_i.$$

The retained space discretisation method is the second order finite volume method thus it allows to handle arbitrary geometries and avoid possible metric singularities. The Finite Volume method is especially adapted to Cartesian grids where, for conforming meshes, it is equivalent to a central difference scheme. We refer to the work of Eymard and co., [36], for comparisons with other discretisation techniques (finite difference schemes, mixed finite element scheme). FV requires no additional treatment on a composite grid and it is also used in elsA, the Onera's software for aerodynamic simulations. In the finite volume method (McDonald 1971), the integral formulation of the conservation laws is discretised directly in the physical space.

In our work, the term "finite volume" will be associated to a small volume surrounding node points on a mesh. The unknowns are average values over the discretisation cells, which are assigned to the centre of these cells. In one dimension (see fig.1.1), each cell Ω_i is defined as $\Omega_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ where $x_{i-\frac{1}{2}} = \frac{x_{i-1}+x_i}{2}$ and $x_{i+\frac{1}{2}} = \frac{x_i+x_{i+1}}{2}$ are the cell interfaces.

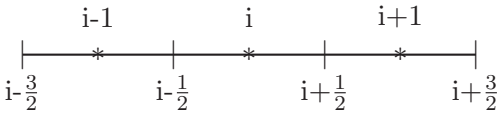


Figure 1.1: Finite Volumes 1D

The integral formulation of the conservative system of equations (1.7) over the computational domain Ω is:

$$\int_\Omega U_t \Phi d_\Omega + \int_\Omega (\nabla \cdot \Psi_{conv}(U) + \nabla \cdot \Psi_{vis}(U, \nabla U)) d_\Omega = 0.$$

This is equivalent to solving on each cell Ω_i:

$$\int_{\Omega_i} U_t d_{\Omega_i} + \int_{\Omega_i} (\nabla \cdot \Psi_{conv}(U) + \nabla \cdot \Psi_{vis}(U, \nabla U)) d_{\Omega_i} = 0,$$

with the following condition of conservativity: the flux leaving the cell i equals the opposite value of the flux entering in the next cell.

First, we consider a semi-discrete scheme. Applying the Stokes theorem, volume integrals that contain a divergence term are converted to surface integrals:

$$\frac{d}{dt} \int_{\Omega_i} U d_\Omega + \int_{\partial\Omega_i} (\Psi_{conv}(U) + \Psi_{vis}(U, \nabla U)).n_{\Omega_i} d_{\partial\Omega_i} = 0,$$

where n_{Ω_i} is the unit outer normal to the cell Ω_i. For a 2D representation, the computation domain consists of a system of elementary cells.

Each basic cell Ω_i is characterized by its volume $\nu(\Omega_i)$ and its surface $\partial\Omega_i$:

$$\nu(\Omega_i) = \int_{\Omega_i} d_{\Omega_i}, \quad \partial\Omega_i = \bigcup_{j=1}^{m} \Gamma_j$$

where Γ_j represents the j-th face of the computational domain and m the number of domain faces, considered in the structured mesh. The FV method can handle any type of mesh, but, in this work only Cartesian meshes are considered, meaning that $m = 4$ for a 2D computation. For a basic cell of the computation domain, the system can be written as:

$$\frac{d}{dt} \int_{\Omega_i} U d_\Omega + \sum_{\Gamma \in \partial\Omega_i} \int_\Gamma (\Psi_{conv}(U) + \Psi_{vis}(U, \nabla U)).n_{\Omega_i,\Gamma} d_\Gamma = 0 \tag{1.8}$$

where $n_{\Omega_i,\Gamma}$ is the unit normal of Γ external to Ω_i. By defining successively:

- the average value of the vector of conservative variables U in the cell Ω_i:

$$\bar{U}_{\Omega_i}(t) = \frac{1}{\nu(\Omega_i)} \int_{\Omega_i} U(t) d_{\Omega_i},$$

- the balance of the convective and diffusive fluxes through the interface Γ of the cell Ω_i :

$$\bar{\Psi}_\Gamma(t) = \int_\Gamma (\Psi_{conv}(U) + \Psi_{vis}(U)).n_{\Omega_i,\Gamma} d\Gamma,$$

and applying the mean value theorem for integration, the system (1.8) can be written as a semi-discretised in space system of ordinary differential equations:

$$\frac{d}{dt}(\nu(\Omega_i)\bar{U}_{\Omega_i}(t)) = -\sum_{\Gamma \subset \partial\Omega_i} \bar{\Psi}_\Gamma(t), \tag{1.9}$$

We introduce then a numerical flux function, Ψ_Γ as a numerical approximation of the exact flux $\bar{\Psi}_\Gamma$, U_i the numerical approximation of the average value $\bar{U}_{\Omega_i}$ and ν_i the cell volume $\nu(\Omega_i)$. After discretisation, the system to solve on a Cartesian mesh becomes a solvable system of differential equations:

$$\frac{d}{dt}(U_i(t)) = -\frac{1}{\nu_i} \sum_{j=1}^{m} \Psi_{\Gamma_j}(t).n_{\Gamma_j}. \tag{1.10}$$

When applied to rectangular meshes, the FV formulation is close to the one found using a finite difference method. The FV method is very flexible since it can be applied on an arbitrary mesh.

1.2.1 Space discretisation

The space discretisation is achieved with finite volumes on Cartesian grids and its goal is to approximate the physical fluxes by numerical fluxes. The three-dimensional computational domain is discretised with N nodes x_i in the x-direction ($x_0 < x_1 < ... < x_{n-2} < x_{n-1}$), M nodes in the y-direction ($y_0 < y_1 < ... < y_{m-2} < y_{m-1}$) and P nodes in the z-direction ($z_0 < z_1 < ... < z_{p-2} < z_{p-1}$). The space steps are the cell lengths $\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}, 0 \leq i < N$, respectively $\Delta y_j = y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}, 0 \leq j < M$ and $\Delta z_k = z_{k+\frac{1}{2}} - z_{k-\frac{1}{2}}, 0 \leq k < P$. The resulting mesh can be uniform (equidistant space steps per direction) or non-uniform (space steps of different sizes in one or more than one direction) which is more often found in practice.

The domain Ω is decomposed in elementary volumes denoted Ω_i as follows:

$$\Omega = \sum_{i=1}^{N \times M \times P} \Omega_i.$$

We integrate the conservation laws on each elementary cell $\Omega_i = [x_{i-1/2}, x_{i+1/2}] \times [y_{i-1/2}, y_{i+1/2}] \times [z_{i-1/2}, z_{i+1/2}]$. Let U_i^n be the approximated value of $\bar{U}_{\Omega_i}^n$ on the cell at time $t = n\Delta t$. For an explicit scheme, at each instant t, the system (1.10) becomes:

$$\begin{aligned} \frac{\partial U_i^n}{\partial t} = \quad & -\tfrac{1}{\nu_i} \quad (F^n_{Euler_{i+1/2}} - F^n_{Euler_{i-1/2}} + G^n_{Euler_{i+1/2}} - G^n_{Euler_{i-1/2}} \\ & + H^n_{Euler_{i+1/2}} - H^n_{Euler_{i-1/2}} \\ & + F^n_{vis_{i+1/2}} - F^n_{vis_{i-1/2}} + G^n_{vis_{i+1/2}} - G^n_{vis_{i-1/2}} + H^n_{vis_{i+1/2}} - H^n_{vis_{i-1/2}}). \end{aligned}$$

The quantity $F^n_{*_{i+1/2}}$ (respectively $G^n_{*_{i+1/2}}, H^n_{*_{i+1/2}}$) is an approximation of the physical flux $F_*(U)$ (respectively $G_*(U), H_*(U)$) on the interface $x_{i+1/2}$ (respectively $y_{i+1/2}, z_{i+1/2}$) called the

numerical flux on the interface. In the simplest case, the numerical flux on the interface depends only on the neighbour values. The way of approximating the numerical flux as a function of the discrete unknown gives the numerical scheme.

The space discretisation is the same as computing the flux balance on one elementary cell. This computation involves summing the interface contributions surrounding the cell. For the scheme to remain conservative, for any interface $i + \frac{1}{2}$, the flux leaving the cell i equals the opposite value of the flux entering in the cell $i+1$. The accuracy of a numerical approximation is directly dependent on the mesh size: the closer the discretised space is to the continuum, the better the approximation of the numerical scheme. For complex geometries, the solution also depends on the shape of the mesh. Another decisive factor in the choice of the size of the mesh is the Reynolds number. It measures the ratio between the inertia terms and the viscous terms that can be drawn into the truncation errors of the inertia terms.

The choice of the Cartesian mesh used in this study will influence the solution especially in cases of shocks where two sub-domains split exactly in the shock region.

1.2.2 Space-Time Boundary Conditions

Solving the system requires the definition of the conservative variables at the initial time (Cauchy problem) and on the boundary of the discrete domain in order to have a well-posed problem. In our work, the computational domain Ω, seen on fig.1.2, comprises cells (denoted fictitious cells) in which values are determined so as to satisfy boundary conditions. Note that the equations are not solved in these fictitious cells. Thus, fictitious data values saved in fictitious cells play the role of boundary conditions. These conditions should respect the fundamental physical laws, the well-posedness of the resulting coupled problem and its consistency with the original problem.

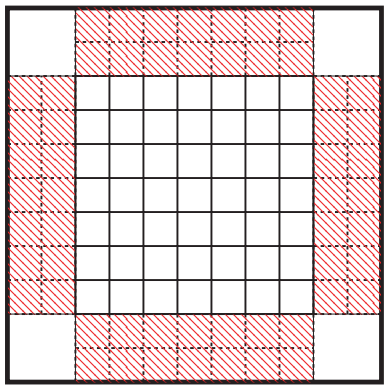


Figure 1.2: 2D Cartesian computational domain Ω. It contains fictitious cells (red) used to save boundary conditions. One layer is sufficient in the case of the Euler system of equations.

When we are talking about the well-posedness of a problem we refer to it in the sense of Hadamard. A problem is well-posed if it fulfils three fundamental properties:

- the problem admits a solution,
- the solution is unique,
- the solution behaviour changes continuously with the initial data.

In the continuous steady case, the time depending term $\frac{\partial}{\partial t}$ disappears and there is no need for initial values. Yet, in order to simulate the steady case we solve a pseudo-unsteady problem, meaning that we introduce a fictitious time term. In this case the convergence depends on the choice of the initial solution: the closer this initial guess is to the final solution, the faster the convergence.

For the unsteady case we need the initial values of $\rho, \rho u, \rho v, \rho w$ and ρE at initial time and in each cell of the field. Usually, the velocities u, v, w are deduced from a given vorticity. We obtain a so-called Cauchy problem:

$$\text{Find U, the solution of} \begin{cases} (1.2) \\ \rho(x,y,z,0) \quad = \rho_0(x,y,z) \\ (\rho u)(x,y,z,0) = (\rho u)_0(x,y,z) \\ (\rho v)(x,y,z,0) = (\rho v)_0(x,y,z) \\ (\rho w)(x,y,z,0) = (\rho w)_0(x,y,z) \\ (\rho E)(x,y,z,0) = (\rho E)_0(x,y,z) \end{cases}, \tag{1.11}$$

where $\rho_0, (\rho u)_0, (\rho v)_0, (\rho w)_0$ and $(\rho E)_0$ are given functions from $\mathbb{R}^3$ to $\mathbb{R}$. Note that the initial solution is defined only inside Ω. When seeking a solution in a bounded domain Ω, it seems natural to add boundary conditions to the Cauchy problem, and then to change the third property in the definition of a well-posed problem into: *the solution behaviour changes continuously with the initial and boundary conditions.*

What are the associated initial and/or boundary conditions? and *How many conditions are to be imposed at a given boundary?* become two of the most common questions one researcher must investigate. In the case of Euler and Navier–Stokes systems of equations, one can find answers in the book of Hirsch [57]. We answer briefly these questions before considering any non-parallel or parallel implementation.

The number of boundary conditions depends on the type of the system to solve. The unsteady Navier–Stokes system of equations can be essentially parabolic in time and space or parabolic – hyperbolic. The steady Navier—Stokes system of equations is elliptic-hyperbolic, elliptic for subsonic flows and hyperbolic for supersonic flows. The Euler system of equation is always hyperbolic, independently of the flow regime. For a problem to be well-posed, the boundary conditions need to be known at each time iteration. Thus, the total number of boundary conditions to be imposed must be equal to the total number of eigenvalues or the number of characteristics entering the domain. To fulfil the well-posedness condition beside boundary conditions defined from outside the domain or physical boundary conditions, one must add boundary conditions defined inside the numerical domain or numerical boundary conditions. The number of physical conditions to be imposed is given by the number of negative eigenvalues of the associated flux matrix, $\Psi_{Euler} \cdot n$, where n is the outward normal vector.

During a computation we can deal with two main types of boundary: a) permeable or b) wall boundary.

a) Permeable boundary

The inflow and outflow boundary conditions are imposed to model a permeable boundary. We present very briefly the main lines of the conducted analysis to find out the number of physical boundaries conditions necessary to find a unique solution for the 3D Euler system of equations (see [57] for details).

The Euler system of equations (1.4) can be written in the following quasi-linear form:

$$U_t + A(U)U_x + B(U)U_y + C(U)U_z = 0, \tag{1.12}$$

where A, B, C are the Jacobian matrices $A(U) = \dfrac{\partial F}{\partial U}, B(U) = \dfrac{\partial G}{\partial U}, G(U) = \dfrac{\partial F}{\partial U}$. The associated eigenvalues to $A(U)\dot{n}_x + B(U)\dot{n}_y + C(U)\dot{n}_z$ can be defined as follows:

- $\lambda^1 = \vec{u}.\vec{n} - a$,
- $\lambda^2 = \vec{u}.\vec{n}$,
- $\lambda^3 = \vec{u}.\vec{n}$,

- $\lambda^4 = \vec{u}.\vec{n}$,
- $\lambda^5 = \vec{u}.\vec{n} + a$,

where $\vec{n}$ is the unit normal vector, external to the computational domain and a the sound speed. Since there are only three different eigenvalues in each direction, we study locally how the propagation of information behaves at a boundary. We can identify three characteristic relations:

- C^- associated to λ^1,
- C^0 associated to λ^2, λ^3 and λ^4,
- C^+ associated to λ^3.

We note $M_a = \frac{|\vec{u}.\vec{n}|}{a}$ the directional Mach number. Let $\vec{s}_n$ be the normal displacement velocity of the boundary and M_{bnd}, a relative normal Mach number defined as:

$$M_{bnd} = \frac{\vec{u}.\vec{n} - \vec{s}_n}{a}$$

The Mach number allows us to separate the subsonic regime $M_{bnd} < 1$ from the supersonic regime $M_{bnd} > 1$. The values of $\vec{s}_n$ are imposed by the flow regime and allow to solve a Riemann problem at the boundary. At a permeable boundary, we identify four possible cases presented on table 1.1.

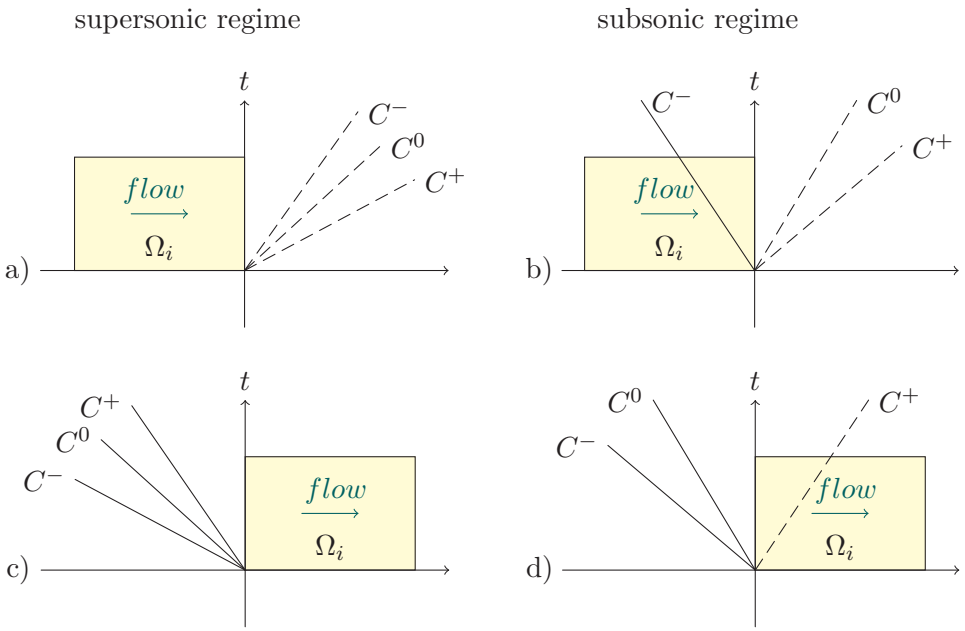


Figure 1.3: Riemann problem for each boundary type

Supposing we are interested in modelling a portion of a passing fluid. The information has to be propagated from inside the domain to outside (fig.1.3.a and fig.1.3.b). It is the so-called outflow (outlet, outgoing) boundary type. On the opposite, an inflow boundary condition (fig.1.3.c and fig.1.3.d) is defined as an information that propagates from outside toward the inside of the domain.

Supersonic regime, $|M_{bnd}| > 1$

- Outflow boundary, $\lambda^i > s_n$, for any i in $\{1, 2, 3, 4, 5\}$. At supersonic outflow (fig.1.3.a) all eigenvalues are positive, they exit the computational domain and no physical conditions have to be given, all boundary conditions need to be numerically defined by the interior flow.

- Inflow boundary, $\lambda^i < s_n, \forall i \in \{1,2,3,4,5\}$. For a supersonic regime (fig.1.3.c) all eigenvalues are negative and consequently all five boundary conditions need to be physically imposed.

Subsonic regime, $|M_{bnd}| < 1$

- Outflow boundary, $\lambda^i > s_n$, for any i in $\{1,2,3,4,5\}$. At subsonic outflow (fig.1.3.b) the normal is pointing toward the interior of the domain and so four eigenvalues are positive and need to be defined as numerical boundary conditions (exit the domain). The fifth eigenvalue is negative and need to be associated to a physical boundary condition (enter the domain).
- Inflow boundary, $\lambda^i < s_n, \forall i \in \{1,2,3,4,5\}$. For a subsonic inflow (fig.1.3.d) the first eigenvalue ($\lambda^1 = \vec{u}.n - a$) is positive and the other four are negative. Therefore, four physical boundary conditions have to be fixed, the fifth one has to be numerically computed.

For the three dimensional Euler equations we centralise these boundary conditions in table 1.1.

Table 1.1: Boundary type and necessary physical boundary conditions for Euler

boundary type	inflow		outflow	
regime	supersonic	subsonic	supersonic	subsonic
relative Mach number	$M_{bnd} < -1$	$-1 < M_{bnd} < 0$	$M_{bnd} > 1$	$0 < M_{bnd} < 1$
number of boundary conditions	5	4	0	1

For the three dimensional Navier–Stokes equations we can refer to the study of L. Halpern [51] and J. Nordstrom and M. Svard [78]. Halpern uses the Fourier transform techniques to set artificial boundary conditions for incompletely parabolic perturbations of hyperbolic systems. J. Nordstrom and M. Svard propose a technique based on the energy method to find the necessary number of boundary conditions, necessary for the well-posedness of the problem. We present their results on the necessary physical conditions in table 1.2.

Table 1.2: Boundary type and necessary physical conditions for Navier–Stokes

boundary type	inflow		outflow	
regime	supersonic	subsonic	supersonic	subsonic
relative Mach number	$M_{bnd} < -1$	$-1 < M_{bnd} < 0$	$M_{bnd} > 1$	$0 < M_{bnd} < 1$
number of boundary conditions	5	5	4	4

In our study we refer to the outflow boundary condition as the transmission boundary condition. They can be treated differently and so we can find them in the literature under different names: open-end boundary condition, far-field boundary condition, radiation boundary condition, transparent or non-reflecting boundary condition. For an inflow boundary we can assume that we have a given velocity profile at the boundary. This leads to an inhomogeneous Dirichlet Condition, which is also the inhomogeneous version of the no-slip condition.

When viscous fluxes are involved, the inflow condition is imposed as a Dirichlet condition, since all conditions are of physical type. At the outflow we impose a zero normal at the interface.

b) Wall boundary

Specific cases of mobile or mixed walls are differently treated, but not studied in this work. Here, we treat only the solid fixed wall. When a boundary of the domain is a fixed wall, the normal velocity is zero $\vec{u} \cdot n$, no mass or convective flux can penetrate the solid. There is no information propagating from outside or inside the domain, but one characteristic, C^0 has a positive sign. Two numerical boundary conditions and three physical ones need to be imposed. In theory, since the velocity profile is linear close to the wall, we can discretise the velocity gradient without any wall boundary conditions. The treatment is different according to the nature of the fluid, inviscid or viscous. For an inviscid fluid coupled with a material wall constituting a slip surface we impose the no-slip condition:

$$(\vec{u} - \vec{s}_w).\vec{n} = 0,$$

where $\vec{s}_w$ indicates the displacement velocity of the boundary and $\vec{n}$ the unit normal vector.

For dissipative wave problems the procedure is considerably more complicated, a thermal condition is added to take into account the temperature or the heat flux at the wall. Moreover, most industrial applications depends on a high Reynolds number. The viscous terms are multiplied by the inverse of the Reynolds number and consequently the system will be dominated by the convective terms. From this property we can deduce that using the same outflow and inflow boundary and initial conditions for the Navier–Stokes equations as for the Euler system of equations is a choice that will not degrade the solution. Of course, the viscosity will play an important role at the wall boundary.

What about the definition of a numerical boundary condition? In most of the presented cases, for the problem to be well-posed we need to numerically define a boundary condition. The simplest way is to extrapolate it from the inside point to the next surface point.

In our simulations the boundary problem is intrinsically solved through the numerical fluxes schemes. The boundary is always defined inside fictitious cells. At the beginning of each iteration the conservative values at the boundary are either imposed as a Dirichlet condition or computed from the closest neighbour at the boundary. The numerical schemes to compute the convective fluxes are constructed so as to differentiate and treat the physical and the numerical boundary conditions. The viscous flow is computed with a Dirichlet type boundary conditions at the inflow (all conditions are physical table 1.2) and a zero normal to the boundary at the outflow.

Let $U_{bnd} = (\rho_{bnd}, (\rho\vec{u})_{bnd}, (\rho E)_{bnd})^T$ be the unknown vector of conservative values at the boundary. We separate the conditions in three types: i) reflective, ii) inflow, and iii) transmissive boundary conditions.

i) The **reflective boundary** condition (see fig.1.4) is defined as:

$$\begin{cases} \rho_{bnd} & = & \rho_{ngh} \\ (\rho\vec{u})_{bnd} \cdot \vec{n} & = & -(\rho\vec{u})_{ngh} \cdot \vec{n} \\ (\rho E)_{bnd} & = & (\rho E)_{ngh} \end{cases},$$

where $U_{ngh} = (\rho_{ngh}, (\rho\vec{u})_{ngh}, (\rho E)_{ngh})^T$ is the vector of conservative values on the neighbour cell and $\vec{n}$ the external unit normal vector. From the physically point of view we can translate this boundary in a fixed, reflective impermeable wall.

ii) At **inflow** we impose a Dirichlet type boundary condition (left on fig.1.5), the value of U on the boundary cell equals a given value:

$$U_{bnd} = U_{dir},$$

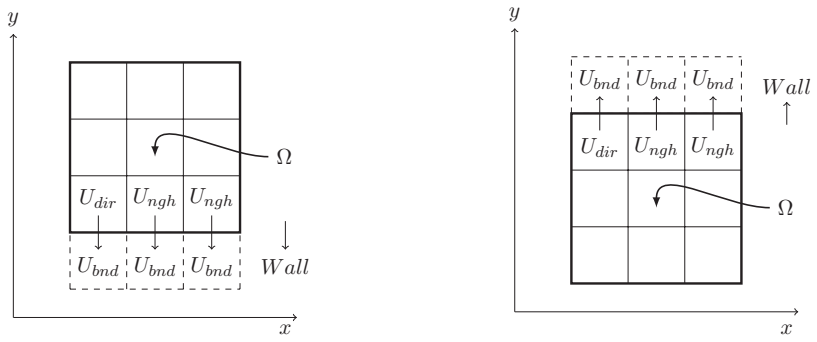


Figure 1.4: 2D representation of a reflexive boundary condition

where $U_{dir} = (\rho_{dir}, (\rho u)_{dir}, (\rho v)_{dir}, (\rho w)_{dir}, (\rho E)_{dir})^T$ is the vector of conservative values arising from physical boundary conditions.

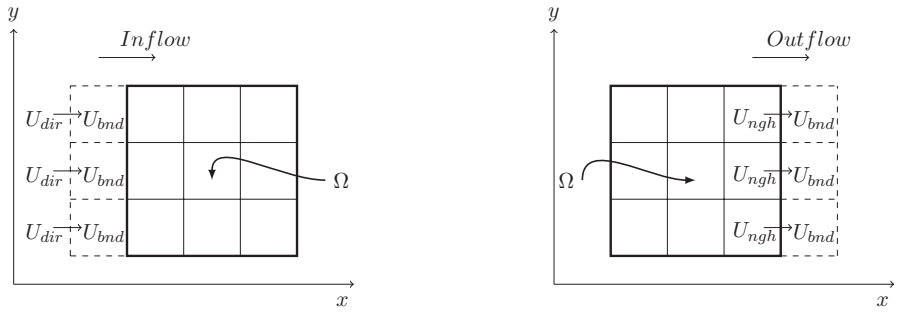


Figure 1.5: 2D representation of a Dirichlet boundary condition (left) and a transmission boundary condition (right).

For the Euler system of equations to be well-posed only the velocity vector and the density are needed (see [85]).

iii) The simplest representation of a numerical **transmissive boundary** condition (right on fig.1.5) is to suppose that the value of U on the limit equals the value of the neighbour cell:

$$U_{bnd} = U_{ngh},$$

where U_* is the vector of conservatives values $U_* = (\rho_*, (\rho u)_*, (\rho v)_*, (\rho w)_*, (\rho E)_*)^T$. This boundary condition is enough for flows with small amplitude fluctuations near the boundary. Unfortunately, it becomes insufficient for large amplitude disturbances at the boundary. This phenomena appears especially for unsteady flows, for example in the case of mixing layer. In this case, for a subsonic outflow only one physical condition must be imposed: the value of the pressure p for a non-reflecting boundary. In our work, when large amplitudes appear at the boundary we choose to absorb these amplitudes by adding sponge regions (fringe regions, absorbing layers, buffer zones) to our computational domain. The sponge region absorbs the undesired waves, but increases the computational cost. Other techniques can also be costly, and they are problem oriented. We can refer to the literature [23, 51, 52, 54, 85] for many attempts to correctly model this kind of boundary. T.J. Poinsot and S.K. Lele [85] conducted a study based on the Navier–Stokes characteristic boundary conditions to identify the viscous conditions for viscous flows. In 2004, Tim Colonius [23] discussed on the evolution of these techniques and reviewed the existing literature. Finding the proper transmission boundary remains an open question, especially for viscous flows.

Interface boundary conditions

Splitting the initial domain into several sub-domains adds interface boundaries which are artificial permeable boundary conditions on the interface between two sub-domains. The interface boundaries can be of several types, as presented on fig.1.6 and fig.1.7, depending on the type of mesh: covered boundary when two sub-domains overlap and so the boundary of one is entirely immersed in the neighbour sub-domain; non-coincident (partially or totally) adjacent boundary when only the interfaces of two sets of cells belong to both sub-domains and they do not necessarily match; coincident adjacent boundary where all grid nodes are coincident with those of the neighbouring sub-domain. In our case the use of fictitious cells imposes a covered boundary, an overlap, but the node grids can be coincident or non-coincident. A non-coincident mesh implies passing interpolated values from on sub-domain to the neighbouring sub-domain. The same technique on the same data in the right and left sub-domain is applied. For Cartesian meshes this approach is conservative if the interfaces are non-matching. For overlapping domains, the fluxes are computed directly on the interface, possible without using the ghost cells (for overlap greater than half stencil size).

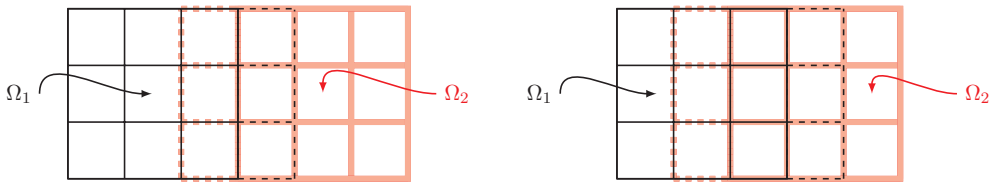


Figure 1.6: 2D coincident interface boundary with coincident adjacent boundary (left) and overlap boundary (right).

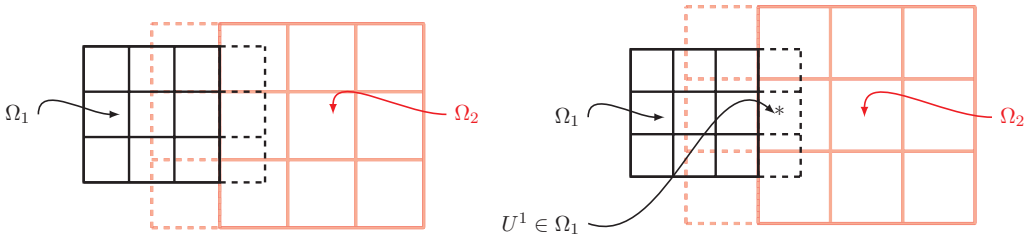


Figure 1.7: 2D partially (left) and totally (right) non-coincident interface boundary.

Lagrange polynomial

When dealing with non-coincident interface boundary a Lagrange polynomial method has been chosen to find the necessary interpolated value to communicate from one sub-domain to another. In each space direction the projection has the same order as the order of the space discretisation thus avoiding different accuracy errors. For a second order space discretisation method we use second order Lagrange polynomials in each direction as presented on fig.1.8.

We present the main steps to find the conservative values to communicate at point (x^1, y^1) on

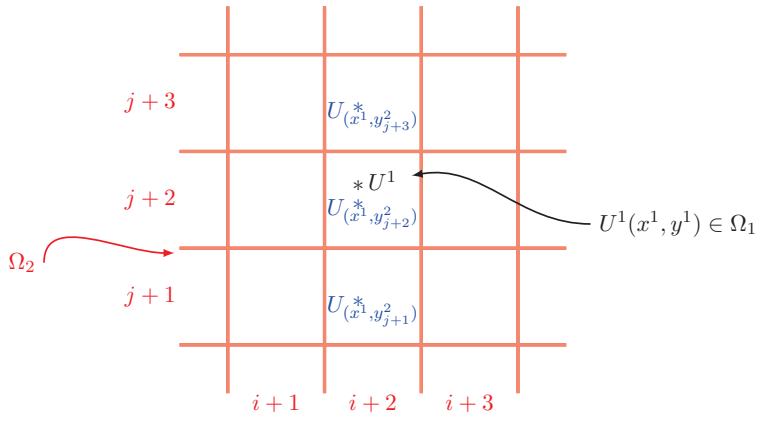


Figure 1.8: Zoom over fig.1.7(right) to highlight the necessarily mesh points of Ω_2 to interpolate the value of $U^1(x^1, y^1) \in \Omega_1$.

fig.1.8:

1. Find the necessary stencil point to compute the intermediate values $U_{(x^1,y^2_{j+1})}$, $U_{(x^1,y^2_{j+2})}$, $U_{(x^1,y^2_{j+3})}$ (blue on fig.1.8). Each of them is computed using the one dimensional second order Lagrange interpolate formula, a linear combination of the Lagrange basis polynomials associated with two down values and one upper value.

$$U(x^1, y^2_*) = \sum_{k=i+1}^{i+3} U^2(x^2_k, y^2_*) l_k(x^1),$$

where $*$ stands for $j+1, j+2$ and $j+3$, $U^2(x^2_k, y^2_*)$ are the vectors of conservative values in Ω_2 computed at positions x^2_k, y^2_*, and l_k are the Lagrange basis polynomials defined as:

$$l_k(x^1) = \prod_{i+1 \le m \le i+3, m \ne k} \frac{x^1 - x^2_m}{x^2_k - x^2_m}, \quad i+1 \le k \le i+3.$$

2. Compute U^1 as a Lagrange interpolation of points $\{(x^1, y^2_{j+1}), (x^1, y^2_{j+2}), (x^1, y^2_{j+3})\}$:

$$U^1(x^1, y^1) = \sum_{k=j+1}^{j+3} U(x^1, y^2_k) l_k(y^1).$$

We note that the same interpolation is used when non-coincident boundary in time appears in the Schwarz Waveform Relaxation Method.

Periodic boundary conditions

Two types of periodic boundary can be defined: periodic by translation or periodic by rotation. For periodic flows by translation, the boundary condition is actually an interface condition and the treatment is consequently the same. For periodic flows by rotation we apply

a rotation matrix to the interface condition, noted $R(\theta)$, of angle of periodicity θ. The rotation matrix is defined following each axis as:

$$R(\theta) = R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix},$$

$$R(\theta) = R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix},$$

$$R(\theta) = R_z(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

1.2.3 Development of the numerical Euler flux

A wide variety of techniques to evaluate the numerical schemes (both convective and diffusive) can be found in literature. Because of their different behaviours and, as seen in section 1.2.2, different number of needed boundary conditions, we treat separately the convective fluxes and the viscous fluxes. The aim of this section is to review some of the most popular approximate schemes of the Euler flux, to highlight their advantages and drawbacks, and then to choose the appropriate one for our applications. We choose a dimensional splitting or fractional method to numerically solve the multi-dimensional Euler system, *i.e.*, we solve a one-dimensional method in each coordinate direction. The finite-volume technique will remain consistent, but there will be some loss of precision in solving the discontinuities that are oblique to grid faces.

In the following, we shall only refer to the x-direction numerical scheme noting that the y-direction and the z-direction are similarly solved. We shall focus on solving the Euler system of equations (1.4) in $x-$direction which is of hyperbolic type with a finite volume method. We solve the following Riemann problem associated to the Euler system of equations to find an exact solution:

$$\begin{cases} U_t + F_{Euler}(U)_x = 0, \\ U(x,0) = U_0(x) = \begin{cases} U_L \text{ if } x < x_0, \ \ x \in [x_L, x_R] \\ U_R \text{ if } x > x_0, \end{cases} \end{cases} \tag{1.13}$$

where U_L and U_R are two vectors of constant values. Two different values for U_L and U_R imply a discontinuity in $x = x_0$. In practice, we consider that x is contained by the interval $[x_L, x_R]$ around the point x_0. From a physical point of view, the Riemann problem associated to the Euler systems of equations is a generalisation of the shock tube problem. A shock tube problem consists in a tube containing two gazes with different properties (density, pressure, velocity...) that are initially separated by a diaphragm. The sudden removal of the diaphragm at time $t = 0$ creates a discontinuity between the two gases. The resulting waves propagate in the tube and can be rarefaction waves (fig.1.9.b, fig.1.9.c), contact discontinuities (fig.1.9.d) or shock discontinuities (fig.1.9.a). A shock wave is a transition layer that appears when the density increases and the velocity decreases very suddenly. A rarefaction wave occurs where the fluid is more rarefied as the density decreases. Contact discontinuities are surfaces that separate zones of different density and temperature.

The values of the density and the internal energy change discontinuously across the contact surface from the constant value at the left of the contact to the constant value at the right of the contact.

We note $\lambda_1 < \lambda_2 < ... < \lambda_m$, the eigenvalues of the linearised Riemann problem of size m, the length of the conservative vector of unknowns ($m = 4$ for a 2D problem, $m = 5$ for a 3D problem). There are $m-$waves associated to the characteristics of the eigenvectors

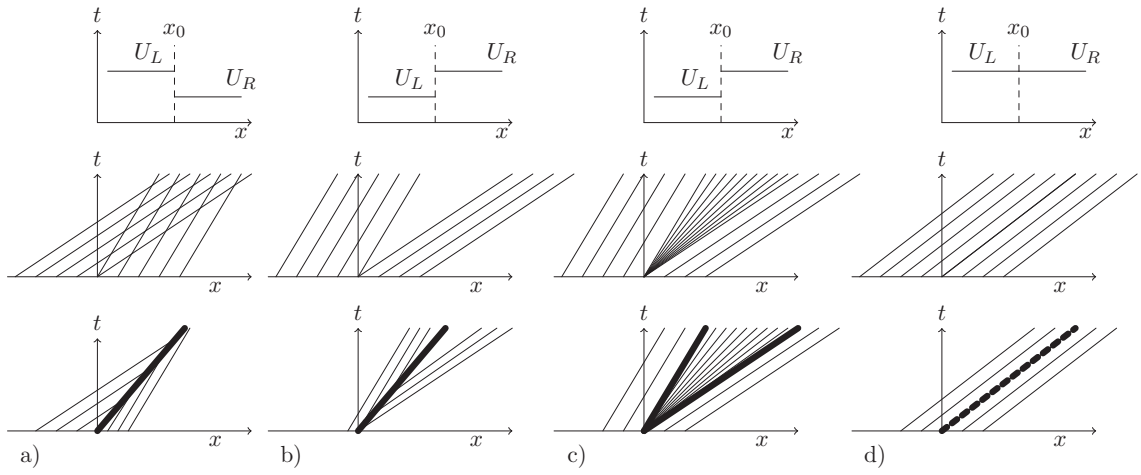


Figure 1.9: Initial solutions and Riemann problem solutions on $x-t$ plane: a) shock , b) rarefaction shock, c) rarefaction wave, d) contact discontinuity.

$K^{(i)},\ i = 1, 2, ..., m$. The solution is given by the different states that a wave can take before, between and after the characteristics. U_L is the solution on the left of the first characteristic, U_R is the solution on the right of the last characteristic corresponding to λ_m. We can express these values as linear combinations of the eigenvectors:

$$U_L = \sum_{i=1}^{m} \alpha_i K^{(i)}, \quad U_R = \sum_{i=1}^{m} \beta_i K^{(i)},$$

with α_i and β_i, $i = 1, .., m$ constant coefficients. The complete solution depends only on the similarity variable $\frac{x}{t}$. We can express the solution of the Riemann problem for a given point (x, t) as:

$$U(x,t) = \sum_{i=I+1}^{m} \alpha_i K^{(i)} + \sum_{i=1}^{I} \beta_i K^{(i)},$$

where λ_I is the eigenvalue such that $\lambda_I < \frac{x}{t} < \lambda_{I+1}$ and $x - \lambda_i t > 0$ for each $i < I$.

When solving the 2D Euler system of equations we can observe three regions, separated by the three associated characteristics. The first region is the left one, where the wave has the same values as U_L, the middle one or so-called the star region is denoted by U^* and its value depends on the results of emerging the two initial waves and the right region where the wave has the values of U_R.

It seems that the Euler system of equations can be solved easily by a Riemann solver, but solving iteratively the exact Riemann problem can be very costly, especially in higher dimension. At each step one should compute the eigenvalues, eigenvectors and characteristics. In practice one uses approximate Riemann solvers and thus, computing numerical fluxes by different techniques that can be classified following different criteria:

- the discretisation:
  - — uncentred schemes (upwind or downwind) depend on the direction of the wave propagation: Godunov, Enquist-Osher, Roe, Rusanov, HLL flux of Harten, HLLC flux of Toro, etc.
  - — centred or symmetric schemes are simpler and do not depend on the direction of the wave: Godunov, Lax-Friedrichs, FORCE (First Order Centred Scheme) flux of Toro, MUSCL, etc.
- the level of discretisation:

— first level: flux vector splitting schemes (van Leer, Steger-Warming, Rusanov, etc.),

— high level: flux difference splitting or Godunov-type methods.

Only the most representative schemes in the literature are presented in this work: Godunov, van Leer, HLLC, LLF and AUSM$^+-$up that approximate the Riemann problem. As already mentioned, only 1D analysis is presented since the fluxes are separated by directions and each one can be solved similarly.

The Godunov Flux on the linearised system

Godunov proposed a first numerical scheme based on either the exact or the approximate solutions of the Riemann problem that solves a non-linear system of equations. The solution is considered as piecewise constant over each mesh cell at a fixed time. Godunov proposes then to use the solution of the Riemann problem along the time axis to define the inter-cells fluxes. System (1.13) is to be solved. The Euler equations are rewritten in conservative form, using the solution of the local Riemann problem on each cell:

$$U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x}(F_{Euler_{i+1/2}} - F_{Euler_{i-1/2}}).$$

$F_{Euler_{i+1/2}}$ is the numerical flux in the $x-$direction defined on the left interface of the cell i as:

$$F_{Euler_{i+1/2}} = F_{Euler}(U_{i+1/2}(0)),$$

where $U_{i+1/2}(0)$ is the exact solution $U_{i+1/2}(x/t)$ of the Riemann problem in $x/t = 0$, the time step Δt must satisfy the condition $\Delta t \leq \frac{\Delta x}{S^n_{\max}}$ where $S^n_{\max}$ represents the maximum velocity of the wave found in the domain at time t^n.

One can now define the CFL number as

$$cfl = \frac{\Delta t S^n_{\max}}{\Delta x}, \quad 0 < cfl < 1. \tag{1.14}$$

The CFL condition allows us to keep the consistency of the scheme through the limitation of the time step Δt such that no interaction between one wave issued at one interface and the waves created at the adjacent interfaces is allowed. $U_{i+1/2}(0)$ that can be easily obtained as

$$U_{i+\frac{1}{2}}(x/t) = \sum_{j=1}^{I} \beta_j K^{(j)} + \sum_{j=I+1}^{m} \alpha_j K^{(j)},$$

where $U_i^n = \sum_{j=1}^m \alpha_j K^{(j)}$, $U_{i+1}^n = \sum_{j=1}^m \beta_j K^{(j)}$ are given as a linear combination of the eigenvectors and $1 \leq I \leq m$ is chosen such as $\lambda_i \leq 0$ for $i \leq I$ and $\lambda_i > 0$ for $i > I$.

The discretisation method is the finite volume method and initially we only dispose of the average values of the conservatives quantities on the cells. But now the values of the Euler fluxes on each inter-cell or interface of the computational domain are required. We develop the expressions of the velocity on the interfaces of the cell i

$$U_{i-\frac{1}{2}}(0) = U_i^n + \sum_{j=1}^{I} (\beta_j - \alpha_j) K^{(j)},$$

$$U_{i+\frac{1}{2}}(0) = U_{i+1}^n - \sum_{j=1}^{I} (\beta_j - \alpha_j) K^{(j)},$$

in a way that they can compute the value of the vector U on the interface $i + \frac{1}{2}$

$$U_{i+\frac{1}{2}}(0) = \frac{1}{2}(U_i^n + U_{i+1}^n) - \frac{1}{2} \sum_{j=1}^{I} sign(\lambda_j)(\beta_j - \alpha_j) K^{(j)}.$$

One can then use these expressions and compute the numerical fluxes between two cells:

$$F_{i+\frac{1}{2}} = \frac{1}{2}(F_i^n + F_{i+1}^n)) - \frac{1}{2}\sum_{j=1}^{m} |\lambda_j|(\beta_j - \alpha_j)K^{(j)}.$$

The van Leer numerical flux

In paper [96], van Leer proposes a definition of the flux in each direction as a function of density, sound speed a and Mach number M_a. In the x-direction the convective flux is rewritten:

$$F_{Euler} = F_{Euler}(\rho, a, M_a) = \begin{pmatrix} \rho a M_a \\ \rho a^2(M_a^2 + \frac{1}{\gamma}) \\ 0 \\ 0 \\ \rho a^3 M(\frac{1}{2}M_a^2 + \frac{1}{\gamma-1}) \end{pmatrix} = \begin{pmatrix} f_{mas} \\ f_{mom} \\ f_{ene} \end{pmatrix},$$

where $M_a = \frac{u}{a}$ and γ is the specific-heat ratio. We can decompose the Jacobian matrix as

$$A = A^+ + A^-, \text{ where } A^{\pm} = \frac{\partial F_{Euler}^{\pm}}{\partial U},$$

and

$$F_{Euler}^{\pm} = \pm\frac{1}{4}\rho a(1 \pm M_a)^2 \begin{pmatrix} 1 \\ \frac{2a}{\gamma}(\frac{\gamma-1}{2}M_a \pm 1) \\ v \\ w \\ \frac{2a^2}{\gamma^2-1}(\frac{\gamma-1}{2}M_a \pm 1) + \frac{1}{2}(v^2 + w^2) \end{pmatrix}.$$

The Mach number is defined on the cell interface as a sum of ratio of speed of different flow regime (supersonic, subsonic):

$$M_{ai+1/2} = M_{a_i}^{+} + M_{a_{i+1,}}^{-}$$

where

$$M_a^{\pm} = \begin{cases} \pm\frac{1}{4}(M_a \pm 1)^2 \text{ if } |M_a| \leq 1 \\ \frac{1}{2}(M_a \pm |M|) \text{ if } |M_a| > 1 \end{cases}.$$

The stability condition using the van Leer schemes is given by:

$$cfl = \frac{\Delta t}{\Delta x}(|u| + a) \leq \frac{2\gamma + |M_a|(3-\gamma)}{\gamma + 3}. \tag{1.15}$$

For an ideal gas, where $\gamma = 1.4$ we find $cfl^{\max} = 1$ for $|M_a| = 1$ and $cfl^{\min} = \frac{2\gamma}{\gamma+3} \approx 0.636...$ for $|M_a| = 0$. The stability condition is more restrictive than the Godunov one, where the CFL number is close to the unity. The scheme has better properties around Mach values of 1 and 0, but the flux does not respect contact discontinuities. The van Leer method is part of a class of methods called flux splitting method. The Flux Vector Splitting (FVS) approach is a technique of distinguishing/ determining the direction of the wave, forward or backward. The resulting splitting scheme combining the dimensional splitting with the finite volume approach may be viewed as a predictor-corrector scheme.

The HLLC numerical flux

Harten, Lax and van Leer [56] propose another approach (HLL) to approximately solve the Riemann problem that is very efficient, but correct only for hyperbolic systems of two equations. The main assumption is to consider only two-wave configurations thus ignoring the possible existence of intermediate waves. The assumption is possible when there is no contact

discontinuity. E.F. Toro, M. Spruce and W. Spears [94] extend the procedure to larger systems by restoring the missing contact surface. They call the new scheme HLLC where C stands for Contact.

Recall the 1D Riemann problem (1.13) to solve:

$$\begin{cases} U_t + F_{Euler}(U)_x = 0, \\ U(x,0) = U_0(x) = \begin{cases} U_L \text{ if } x < 0, \\ U_R \text{ if } x > 0, \end{cases} \end{cases} \tag{1.16}$$

where U_L is the solution on the left of the first characteristic, U_R is the solution on the right of the last characteristic. We consider S_L and S_R the fastest signal velocities perturbing the initial data states U_L and U_R, S^* the speed of the middle wave in the star-region.

The HLLC flux takes into account three wave patterns, namely the subsonic flow (found for $S_L \leq 0,\ S_R \geq 0$), the right supersonic flow ($S_L \geq 0$) and the left supersonic flow ($S_R \leq 0$). By applying the Rankine-Hugoniot conditions across each wave of speeds S_L, S^*, S_R one can rewrite the flux in function of the flow regime (subsonic, supersonic) as:

$$F^{hllc}_{Euler_{i+\frac{1}{2}}} = \begin{cases} F_L = F_{Euler}(U_L) & \text{if} \quad 0 \leq S_L, \\ F^*_L = F_L + S_L(U^*_L - U_L) & \text{if} \quad S_L \leq 0 \leq S^*, \\ F^*_R = F_R + S_R(U^*_R - U_R) & \text{if} \quad S^* \leq 0 \leq S_R, \\ F_R = F_{Euler}(U_R) & \text{if} \quad S_R \leq 0, \end{cases}$$

where

$$U^*_k = \rho_k \left(\frac{S_k - u_k}{S_k - S^*}\right) \begin{pmatrix} 1 \\ S^* \\ \frac{E_k}{\rho_k} + (S^* - u_k)[S^* + \frac{p_k}{\rho_k(S_k - u_k)}] \end{pmatrix} \text{ for } k = L, R.$$

We can observe that, the scheme allows the intrinsic computation of the unknown boundary conditions. Multiple ways of providing wave speed estimates for the lower and upper bounds of the wave speeds S_L and S_R are proposed by Toro in [93] and [94]. In our study we shall use the expressions:

$$S_L = \min(u_L - a_L, u_R - a_R), \quad S_R = \max(u_L + a_L, u_R + a_R).$$

The LLF numerical flux

LLF stands for local Lax–Friedrichs, but can also be found in the literature as the Rusanov flux. At each interface of the cell a Riemann problem is solved. We first compute the wave speed estimates S_L and S_R, but then only the upper bound for the absolute values of the characteristic speeds S^+ is considered.

$$S^+ = \max(S_L, S_R).$$

The inter-cell flux is given by:

$$F_{Euler_{i+\frac{1}{2}}} = \frac{1}{2}[F_{Euler}(U_R) + F_{Euler}(U_L)] - S^+(U_R - U_L).$$

We note that the interval $[U_L, U_R]$ is always non-empty. The expression of the inter-cell flux is similar to the HLL one, but the computation is faster because the Rusanov scheme does not involve computations of half-steps values.

The AUSM$^+$-up numerical flux

The Advection Upstream Splitting Method (AUSM), introduced by Liou and Steffen in [70], is a hybrid splitting of flux-difference and flux-vector, benefiting of the accuracy and the simplicity of the splitting methods. The AUSM is based on splitting the Euler flux into convective and pressure fluxes:

$$F_{Euler}(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho u(E+p) \end{pmatrix} = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv \\ \rho uw \\ \rho uH \end{pmatrix} + \begin{pmatrix} 0 \\ p \\ 0 \\ 0 \\ 0 \end{pmatrix} = F^c + F^p,$$

$$G_{Euler}(U) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho v(E+p) \end{pmatrix} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 \\ \rho vw \\ \rho vH \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ p \\ 0 \\ 0 \end{pmatrix} = G^c + G^p,$$

$$H_{Euler}(U) = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho w(E+p) \end{pmatrix} = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 \\ \rho wH \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ p \\ 0 \end{pmatrix} = H^c + H^p.$$

The convective term in the x-direction (similar in the y and $z-$ directions) is rewritten as follows:

$$F^c_{Euler} = \rho u \begin{pmatrix} 1 \\ u \\ v \\ w \\ H \end{pmatrix} = \dot{m}\vec{F}^c_{Euler},$$

where $\dot{m} = \rho u$, H is the enthalpy $H = E + \frac{p}{\rho}$. Therefore, the interface flux $F_{Euler}{}_{i+\frac{1}{2}}$ is computed as the sum between the convective and the pressure flux at the same interface:

$$F_{Euler}{}_{i+\frac{1}{2}} = F^c_{Euler}{}_{i+\frac{1}{2}} + F^p_{Euler}{}_{i+\frac{1}{2}} = \dot{m}_{i+\frac{1}{2}}[\vec{F}^c_{Euler}]_{i+\frac{1}{2}} + + F^p_{Euler}{}_{i+\frac{1}{2}}.$$

The definition of the convective flux gives the main steps of the algorithm. In further papers, Liou [67, 68, 69] improves the original AUSM algorithm by adding important properties to the definition of the convective flux. The later algorithm is designated as AUSM$^+$-up [69] because it incorporates both velocity and pressure terms into AUSM$^+$ (Liou [67]) scheme, the improved AUSM scheme to capture a stationary shock. The aim of AUSM$^+$-up is to be uniformly valid for all speed regimes.

AUSM$^+$-up algorithm

The following steps are valid for any interface of the domain Ω, $i + \frac{1}{2} \overset{\text{noted}}{=} \frac{1}{2}$. The flux computation is similar in each spatial direction. We shall only present its computation in the x-direction.

1. Compute $\dot{m}_{\frac{1}{2}} = u_{\frac{1}{2}}\rho_{\frac{L}{R}} = a_{\frac{1}{2}}M_{a\,\frac{1}{2}}\rho_{\frac{1}{2}}$ where L and R are respectively the left and right indexes of the cells containing a same interface, a is a non-zero constant propagation of sound and M_a is the Mach number. The value of $\rho_{\frac{L}{R}}$ is different from the value at the interface $\rho_{\frac{1}{2}}$ and will be defined below among all other terms.

- $a_{\frac{1}{2}} = \min(a_R, a_L)$
  where $a_L = \dfrac{(a^*)^2}{\max(a^*, |u_L|)}$, $a_R = \dfrac{(a^*)^2}{\max(a^*, |u_R|)}$, $(a^*)^2 = \dfrac{2(\gamma-1)}{\gamma+1}H_t$, a^* is the critical speed of sound evaluated in u_L, H_t is the total enthalpy, $H_t = \dfrac{a^2}{\gamma-1} + \frac{1}{2}u^2$, $u = \vec{u}\cdot\vec{n}$ with $\vec{n}$ the unit normal vector for multi-dimensional flows. In order to properly scale the numerical dissipation with the flow speed we multiply, as in [69], $a_{\frac{1}{2}}$ by a scaling factor f_a ($\tilde{a}_{\frac{1}{2}} = f_a a_{\frac{1}{2}}$).
  $f_a(M_0) = M_0(2 - M_0) \in [0,1]$ where $M_0 = \min(1, \max(\bar{M}_a^2, M_{a\infty}^2))$ and $\bar{M}_a^2 = \dfrac{u_L^2 + u_R^2}{2a_{\frac{1}{2}}^2}$.
- $M_{a\frac{1}{2}} = \mathcal{M}^+_{(4)}(M_{aL}) + \mathcal{M}^-_{(4)}(M_{aR}) - \dfrac{K_p}{f_a}\max(1 - \sigma\bar{M}_a^2, 0)\dfrac{p_R - p_L}{\rho_{\frac{1}{2}} a_{\frac{1}{2}}^2}$,
  where $\rho_{\frac{1}{2}} = (\rho_L + \rho_R)/2$, $0 \le K_p \le 1$, $\sigma \le 1$ and

$$\mathcal{M}^{\pm}_{(4)}(M_a) = \begin{cases} \dfrac{1}{2}(M_a \pm |M_a|) & \text{if } |M_a| \ge 0, \\ \pm\dfrac{1}{4}(M_a \pm 1)^2(1 \mp 16\beta(\pm\dfrac{1}{4}(M_a \pm 1)^2)) & \text{if } |M_a| \le 0. \end{cases}$$

- Then, the mass fluxes are defined as

$$\dot{m}_{\frac{1}{2}} = a_{\frac{1}{2}} M_{a\frac{1}{2}} \begin{cases} \rho_L & \text{if } M_{a\frac{1}{2}} > 0, \\ \rho_R & \text{if } M_{a\frac{1}{2}} \le 0. \end{cases}$$

2. Keeping the same definition as for the mass fluxes computation, we define the pressure fluxes as:

$$p_{\frac{1}{2}} = \mathcal{P}^+_{(5)}(M_{aL})p_L + \mathcal{P}^-_{(5)}(M_{aR})p_R - K_u\mathcal{P}^+_{(5)}\mathcal{P}^-_{(5)}(\rho_L + \rho_R)(f_a a_{1/2})(u_R - u_L)$$

   with $0 \le K_u \le 1$ and

$$\mathcal{P}^{\pm}_{(5)}(M_a) = \begin{cases} \dfrac{1}{M_a}\dfrac{1}{2}(M_a \pm |M_a|) & \text{if } |M_a| \ge 0, \\ \pm\dfrac{1}{4}(M_a \pm 1)^2[(\pm 2 - M_a) \mp 16\alpha M_a(\pm\dfrac{1}{4}(M_a \pm 1)^2)] & \text{if } |M_a| \le 0, \end{cases}$$

   using the parameters: $\alpha = \dfrac{3}{16}(-4 + 5f_a^2) \in \big[-\dfrac{3}{4}, \dfrac{3}{16}\big]$, $\beta = \dfrac{1}{8}$.
3. Finally, the flux becomes

$$F^c_{Euler_{\frac{1}{2}}} = p_{\frac{1}{2}} + \cdot m_{\frac{1}{2}} \begin{cases} F^c_{Euler_L} & \text{if } \cdot m_{\frac{1}{2}} > 0 \\ F^c_{Euler_R} & \text{if } \cdot m_{\frac{1}{2}} \le 0 \end{cases}$$

The basic scheme is obtained for $f_a = 1$. The AUSM$^+$-up was proven to be the best candidate for our explicit and implicit applications and is the only one computed on GPU. In the latest International Conference on Computational Fluid Dynamics (ICCFD8), Moguen and co [74] propose another improvement on the AUSM$^+$-up scheme adding inertia terms in the face velocity expression of Godunov-type schemes. The resulting scheme is called AUSM-IT and allows full Mach number range high quality calculations (especially in cases of Mach numbers smaller than the unity that can impose difficulties for the AUSM$^+$-up scheme).

In order to extend the schemes to second order we use a MUSCL method.

Development of a second order numerical flux using the MUSCL scheme

All presented numerical schemes are of first order in space. In our study, we extend presented numerical schemes to the second order schemes in space using the piecewise linear rebuilding of the approximate solution introduced by van Leer in [95]. MUSCL stands for Monotone Upstream centred Scheme for Conservative Laws or Variable Extrapolation approach and it was used to compute the first high order numerical scheme based on the idea of increasing the precision by increasing the interpolation degree of the unknowns.

Recall the conservative form of the Euler Equations:

$$U_t + F_{Euler}(U)_x + G_{Euler}(U)_y + H_{Euler}(U)_z = 0.$$

The second order, MUSCL approach for computing Euler fluxes is based on data reconstruction and is achieved in three main steps. We only present the three steps for the reduced 1D system, since the flux computation is similar in each direction. The 1D conservative Euler system is:

$$U_t + F_{Euler}(U)_x = 0.$$

MUSCL algorithm in three steps

1. **Local Data Reconstruction**
   Let U_i^n be the data average value in cell $I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$. It is locally replaced by a piece-wise linear function:

   $$U_i(x) = U_i^n + \frac{x - x_i}{\Delta x}\Delta i, \quad x \in I_i,$$

   where Δi is a chosen slope vector of $U_i(x)$ in cell I_i that indicates the variation of U_i variable. The slope limiter prevents numerical oscillations around discontinuities or shocks, but it can diminish the order of the scheme around these regions. $U_i(x)$ has two extreme points $U_i^L = U_i(0) = U_i^n - \frac{1}{2}\Delta i$ and $U_i^R = U_i(\Delta x) = U_i^n + \frac{1}{2}\Delta i$ called *boundary extrapolated values.*

2. **Evolution** of the solution U_i^L, U_i^R by half the full time step, $\frac{\Delta t}{2}$:

   $$\begin{cases} \bar{U}_i^L = U_i^L + \frac{1}{2}\frac{\Delta t}{\Delta x}[F_{Euler}(U_i^L) - F_{Euler}(U_i^R)] \\ \bar{U}_i^R = U_i^R + \frac{1}{2}\frac{\Delta t}{\Delta x}[F_{Euler}(U_i^L) - F_{Euler}(U_i^R)]. \end{cases}$$

3. **Resolution of the Riemann problem**

   $$\begin{cases} \partial_t U_i + \partial_x F_{Euler}(U_i) = 0, \\ \qquad\qquad U_i(x,0) = \begin{cases} \bar{U}_i^R, & x < 0, \\ \bar{U}_i^L, & x > 0, \end{cases} \quad x \in I_i. \end{cases}$$

The choice of the slope limiter must be done very carefully. Its value must be greater than or equal to zero. Note that a limiter equal to zero leads to the loss of one order of accuracy and that any of the proposed limiters (see below) works well for all problems.

Choice of Δi. Define:

$$\Delta_{i+\frac{1}{2}} = U_{i+1}^n - U_i^n \text{ and } \Delta_{i-\frac{1}{2}} = U_i^n - U_{i-1}^n,$$

the interface slope vectors. A possible choice for the slope vector Δ_i is

$$\Delta_i = \frac{1}{2}(1-\omega)\Delta_{i-\frac{1}{2}} + \frac{1}{2}(1+\omega)\Delta_{i+\frac{1}{2}}, \ \omega \in [-1, 1],$$

but, some spurious oscillations can appear in the vicinity of strong gradients. One way to deal with this problem is to use some Total Variation Diminishing (TVD) constraints. It involves replacing the slopes Δ_i by a limited slope $\bar{\Delta}_i$:

$$\bar{\Delta}_i = \begin{cases} \max[0, \min(\beta\Delta_{i-1}, \Delta_{i+1}), \min(\Delta_{i-1}, \beta\Delta_{i+1})], & \Delta_{i+1} > 0, \\ \min[0, \max(\beta\Delta_{i-1}, \Delta_{i+1}), \max(\Delta_{i-1}, \beta\Delta_{i+1})], & \Delta_{i+1} < 0. \end{cases}$$

For particular values of the parameter β we get different limited slopes, already defined in the literature:

- $\beta = 1$ reproduces the MINMOD (or MINBEE) flux limiter,
- $\beta = 2$ reproduces the SUPERBEE flux limiter.

Another way is to find the constant ξ_i so that $\bar{\Delta}_i = \xi_i \Delta_i$ where, for example:

$$\bar{\Delta}_i = \sum_{k=1}^{N} \bar{\Delta}_i^k(\Delta_{i-1/2}^k, \Delta_{i+1/2}^k).$$

This choice leads to a TVD region for $\xi(r)$:

$$\xi(r) = 0 \quad si \quad r \leq 0, \quad 0 \leq \xi(r) \leq \min(\xi^L(r), \xi^R(r)) \; si \; r > 0$$

$$\bar{\Delta}_i = \begin{cases} \xi^L(r) = \frac{2\beta_{i-1/2} r}{1-\omega+(1+\omega)r}, \\ \xi^R(r) = \frac{2\beta_{i+1/2}}{1-\omega+(1+\omega)r}, \\ r \qquad = \frac{\Delta_{i-1/2}}{\Delta_{i+1/2}}, \end{cases}$$

and $\beta_{i-1/2} = \frac{2}{1+c}$, $\beta_{i+1/2} = \frac{2}{1-c}$ where c is the CFL or Courant number of the wave. For

$$\xi_{va}(r) = \begin{cases} 0, & r \leq 0, \\ \min(\frac{r(1+r)}{1+r^2}, \xi_R(r)), & r \geq 0, \end{cases}$$

we obtain the VAN ALBADA limiter, which is less diffusive than other limiters (for example MINMOD) and provide sharper results. We can find many other proposed slope limiters in the literature like the Osher limiter (Chatkravathy and Osher, 1983), the van Leer limiter (van Leer, 1974), monotonized central (MC - van Leer, 1977) and others, but we will limit our analysis to the presented ones as they seems to be the most common, and achieve good results.

1.2.4 Development of the numerical viscous flux

We recall the resulting system to solve on a Cartesian mesh found in section 1.2 with a finite volume method, (1.10). For each cell Ω_i:

$$\frac{d}{dt}(U_i(t)) = -\frac{1}{\nu_i} \sum_{j=1}^{m} \Psi_{\Gamma_j}(t) \cdot n_{\Gamma_j}.$$

Suppose that the 1D finite volume discretisation is as on fig.1.10, and let F_{vis} be the unidirectional flux. In one dimension the system is equivalent to:

$$\frac{d}{dt}(U_i(t)) = -\frac{1}{\Delta x_i}\big(F_{vis}(x_{i+\frac{1}{2}}) - F_{vis}(x_{i-\frac{1}{2}}) + F_{Euler}(x_{i+\frac{1}{2}}) - F_{Euler}(x_{i-\frac{1}{2}})\big).$$

We propose a classical scheme to compute the viscous flux $F_{vis} = (0, \tau, u\tau + q)^t$ with $\tau = -2\lambda\dfrac{\partial u}{\partial x}$. We note $F_{vis}(x_i) = \frac{1}{\Delta x_i}\big(F_{vis}(x_{i+\frac{1}{2}}) - F_{vis}(x_{i-\frac{1}{2}})\big)$.

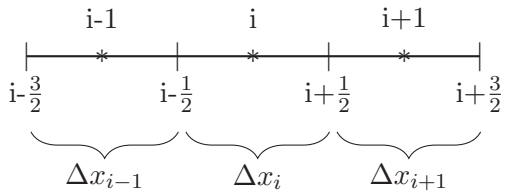


Figure 1.10: Finite Volumes 1D

The fluxes at interfaces are evaluated by central average from the neighbours cells, $F_{vis}(x_{i-\frac{1}{2}}) = \frac{F_{vis}(x_{i-1}) + F_{vis}(x_i)}{2}$ and $F_{vis}(x_{i+\frac{1}{2}}) = \frac{F_{vis}(x_i) + F_{vis}(x_{i+1})}{2}$. Thus, the viscous part in cell Ω_i becomes:

$$F_{vis}(x_i) = \frac{F_{vis}(x_{i+1}) - F_{vis}(x_{i-1})}{2\Delta x_i}.$$

For a 1D domain, if we apply this expression to $F_{vis}(x_i) = (0, \tau, u\tau+q)^t(x_i)$ and compute the velocity gradient using a centred scheme we find the following expressions for the components of U which have the stencils represented on 1.11.

$$\tau(x_i) \simeq -2\lambda \frac{\frac{\partial u}{\partial x}(x_{i+1}) - \frac{\partial u}{\partial x}(x_{i-1})}{2\Delta x_i} \simeq -2\lambda \frac{\frac{u(x_{i+2}) - u(x_i)}{\Delta x_{i+1}} - \frac{u(x_i) - u(x_{i-2})}{\Delta x_{i-1}}}{2\Delta x_i},$$

$$(u\tau + q)(x_i) \simeq \frac{-2\lambda\left[u(x_i+1)\frac{u(x_{i+2}) - u(x_i)}{\Delta x_{i+1}} + u(x_{i-1})\frac{u(x_i) - u(x_{i-2})}{\Delta x_{i-1}}\right]}{2\Delta x_i} + \frac{q(x_{i+1}) - q(x_{i-1})}{2\Delta x_i}.$$

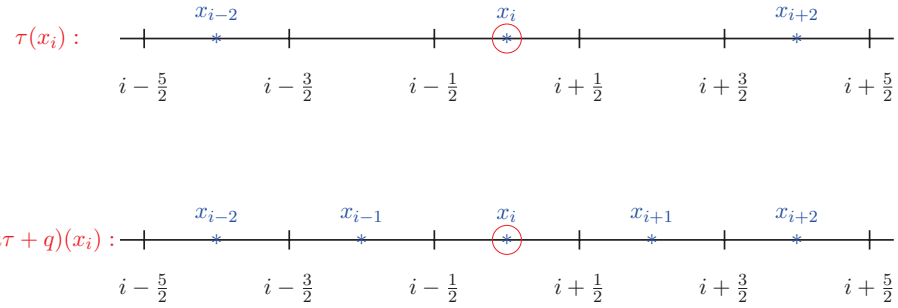


Figure 1.11: Stencils of the diffusive 1D flux

This approach is also used in elsA. The centred character of the scheme is ensured even on the boundaries via the ghost cells. For higher dimensional problems, the computation of the viscous fluxes follows the same reasoning in each spatial direction.

1.3 Explicit and Implicit Time Schemes

After the space discretisation, the unknown of the problem is the solution vector at each cell of the dicretisation as a time function. The spatial discretisation leads to a system of ordinary differential equations in time. The solution is either steady, not changing during the time evolution, either unsteady or varying with the time. We divide the time axis in intervals Δt and compute the field at instant t^n. The time discretisation can be done explicitly or

implicitly. If the fluxes are computed at time t^n, the integration method is called explicit and it can be characterised by the rapidity of the integration of the system of equations on one time step. The matrix of the unknown variables is a diagonal matrix, while the right-hand side is dependent only on previous time flow. This leads to a trivial matrix inversion to find the solution. However, in order to ensure the stability of the integration and convergence, the time step value has to be small as it must verify the Courant-Friedrichs-Lewy (CFL) stability condition. The CFL number is a dimensionless quantity close to the unit, but its value can differ with the explicit scheme, as already seen in the previous sections. Explicit methods adapt very well to unsteady cases, since the physical phenomena can vary even inside very small time intervals. For steady cases, the computation may require a large number of time steps to reach the steady-state solution corresponding to a physical time-independent problem.

If the fluxes are estimated at time t^{n+1} the integration method is called implicit. The matrix to be inverted is not diagonal, but in most cases, it has a rather simple structure, such as block tridiagonal. This discretisation ensures the stability for bigger CFL numbers, but leads to the resolution of an algebraic system at each time step, which can be very expensive in computational cost. This is compensated in many implicit methods, especially in linear problems, by the fact that there is no limitation on the size of the time step. Implicit methods adapt well especially to reach the steady-state cases.

To solve the algebraic systems, there exist two families of methods: direct and iterative. A direct method is characterised by giving the solution in one step, as an iterative method requires many iterative steps. Solving a non-linear system requires an iterative method. Many accelerators of the iterative methods have been studied and we distinguish preconditioning and multi-grid methods. We are interested in both steady and unsteady formulations. In the case of a steady problem we introduce a fictitious time step (or pseudo time integration) and solve the system as the previous unsteady formulation.

Let us denote the semi-discretised system in time of one ODE system as follows:

$$\frac{dU}{dt} + \psi(U) = 0 \quad \text{on} \quad \Omega \times [0, T],$$

where ψ_i, the i-th component of ψ is the sum of convective and diffusive fluxes of U.

1.3.1 First and Second order Explicit Methods

Let us consider the next one dimensional Cauchy problem:

$$\begin{cases} \partial_t U(t,x) = \psi(t, U(t,x)) \ , \ 0 \leq t \leq T \ , \ x \in \Omega \\ U(t^0, x) \ = U_0(x) \end{cases},$$

where ψ is the total flux and $\Omega = [a, b]$ the 1D domain of definition.

We divide the time axis in N intervals Δt: $0 = t^0 < t^1 < .. < t^{N-1} < t^N = T$. We are searching an approximation U^n of U at the instant t^n, $1 \leq n \leq N$. In order to compute U^n, we choose two explicit discretisation methods: The First Order Forward Explicit Euler Method and the Second order Runge Kutta Method.

First order Explicit Euler Method

Named after the Swiss mathematician Leonhard Euler who introduced it in his book *Institutionum calculi integralis, 1768-1770*, the Euler Method is the simplest, first-order numerical method to solve an ODE. The Euler Method can be derived from the Taylor expansion:

$$U(t^n) = U(t^{n-1}) + \Delta t \partial_t U(t^{n-1}) + O(\Delta t^2).$$

We drop the last term in the Taylor expansion and find the first order in time Forward Euler Method :

$$U^n = U^{n-1} + \Delta t \psi(t^{n-1}, U^{n-1}).$$

The Forward Euler method is conditionally stable, under a CFL type condition which depends on the numerical scheme that computes the fluxes. The Backward Euler Method is an implicit method and it requires the knowledge of U at the instant t^{n+1}. It will be treated in the second chapter of this manuscript. The solutions provided with a first order numerical scheme have a low accuracy. In order to develop a more stable approximation of U we use a higher order numerical method.

Second order Runge Kutta Method

The Runge–Kutta (RK) methods are one step methods that can be either explicit or implicit methods. In this work we will only study the second order explicit RK method also called the explicit trapezoidal method, consistent with the presented Finite Volume scheme. A higher order method can be more stable, but they are not more efficient, since they require a larger amount of computations. The second order Runge–Kutta method is a composition of the Euler Method limited to two time levels. It consists on the estimation of the middle of the integration step. We consider $U^{n+\frac{1}{2}}$ the approximated solution of U at the instant $t^{n+\frac{1}{2}}$ computed by the first order forward Euler method:

$$U^{n+\frac{1}{2}} = U^n + \frac{\Delta t}{2}\psi(t, U^n),$$

$$\dot{U}^{n+\frac{1}{2}} = \psi(t + \frac{\Delta t}{2}, U^{n+\frac{1}{2}}),$$

followed by the rebuild of the complete integration based on this first approximation:

$$U^{n+1} = U^n + \Delta t \dot{U}^{n+\frac{1}{2}}.$$

The RK methods are easy to operate with variable time steps. If we consider the evaluation of ψ as a cost measure, we can notice that the cost of a second order RK method is twice the first order Explicit Euler Method, but the error between the approximate solution and the exact one is much smaller. The choice between the two methods is case oriented, and it depends on the required accuracy of the solution. The explicit methods need to satisfy a stability condition, yet, one can use the largest time step satisfying this conditions.

Stability limit for an explicit scheme

We distinguish three kind of stabilities: physical, mathematical and numerical. A physical instability or chaos appears when a small variation inside initial data leads to unpredictable phenomena. The mathematical stability can be seen as a sensibility inside an ill conditioned system: a small variation inside the initial data or parameters leads to different results. The third type of stability, the numerical stability, appears when one simulates phenomena that are physically and mathematically stable, but because of some propagation error an unexpected behaviour occurs. An error propagation can simply arise from the finite precision of computer arithmetic that causes an algorithm to solve a perturbed problem.

To check the mathematical stability of the presented numerical schemes, a von Neumann analysis should be conducted on both explicit Euler method and Runge Kutta method combined with each presented numerical flux. The analysis is quite difficult and the stability conditions can differ from one numerical scheme to another. We will refer to the work of Toro [93] and Hirsch [57] for further details on the von Neumann analysis, and we will consider what seems to

be a practical stability condition. Moreover, we distinguish two mathematical stability criteria: one issued from the Euler system of equations (the convective stability criteria), and one for the diffusive flows. All resulting schemes are conditionally stable under conditions of Courant-Friedrichs-Lewy (CFL) type.

As for example, for the convective CFL number, the expression found solving a 1D Euler system of equations using the Godunov numerical scheme to compute the numerical fluxes (1.14) is:

$$0 < cfl = \frac{a\Delta t_C}{\Delta h} < 1,$$

where Δh is the constant space step, a is the wave propagation speed and Δt_c the time step. We can define the CFL as a ratio of two speeds: the wave propagations speed a and the grid speed $\frac{\Delta t_c}{\Delta h}$ imposed by the discretisation of the computational domain. In practice, we shall impose the CFL number found by computing the Euler fluxes with a FVS (Flux Vector Splitting) scheme, the van Leer numerical scheme (see (1.15)) which is also used by elsA to compute the time step of the simulation:

$$0 < \Delta t_C = cfl\frac{\Delta h}{|U| + a} < 1.$$

The diffusive stability criteria is computed on a centred explicit scheme using the von Neumann method:

$$0 < \Delta t_D = cfl\frac{(\Delta h)^2}{2}\frac{\rho}{\mu}\frac{Pr}{\gamma} < 1,$$

where Pr is the Prandtl number, μ represents the laminar viscosity and Δt_D the time step. The local time step in the explicit case is computed by joining together the results of both convective and diffusive fluxes:

$$\Delta t = \min(\Delta t_C, \Delta t_D), \tag{1.17}$$

for the same characteristic cell length, Δh. In the 2D and 3D cases the same CFL condition is asked to be satisfied in each direction, the global stability condition is obtained by minimizing the directional stability conditions.

The studied implicit schemes are unconditionally stable, meaning there is no analytically imposed restriction over the size of the time step. Yet, the solution must remain into the physical bounds of the problem (kinetic energy for example) which adds another limit to the time step. Thus, in our simulations, a similar relation to the CFL condition is used to ensure numerical convergence.

1.3.2 Second order Implicit Backward Differentiation Methods

In this section we consider another kind of scheme to solve the Navier–Stokes (Euler) system of equations: the implicit scheme. The chosen implicit scheme is a multi-step method, the second order accuracy backward differentiation method (BDF). The scheme is unconditionally stable for a steady case and it must have a fast convergence for very large time steps. For an unsteady case, to ensure the numerical stability, the time step needs to be controlled, chosen carefully and can be limited. The implicit scheme allows longer simulations and is particularly adapted to the steady case. Compared to one-step methods, it usually requires fewer function evaluation per step, but it is usually more expensive.

The Backward Differentiation Methods are numerical methods of integration based on the backward differentiation formulas (BDF). The BDF scheme was first proposed by Curtiss and Hirschfelder en 1951 in [27] as a scheme to solve stiff equations of any degree of accuracy. The BDF gives an approximation of a time derivative t^{n+1} depending on its value at time t^n, by differentiating the polynomial that interpolates the past known values and to set the derivative at t^{n+1} of the right term. They are linear multi-steps methods using the approximate

values $U^n, U^{n-1}, ..$ of U, whose components are the approximated solutions in each node of the discretised space domain. The general form of the k-order BDF method is:

$$\sum_{i=0}^{k} \alpha_i U^{n-i} = -\Delta t \beta_0 \psi(U^n),$$

where $\alpha_0 = 1$. For $\beta_0 = 0$, the resulting scheme is explicit, otherwise it is implicit. The BDF methods are stable up to order 6 (for discussions over the BDF methods and stability study see [1]). In table 1.3, we give the coefficients of the BDF methods up to order 6 for a fixed time step. Beyond sixth order, the absolute stability region of the resulting BDF methods is very small and the methods are useless.

Table 1.3: Coefficients of BDF up to order 6

p	k	β_0	α_0	α_1	α_2	α_3	α_4	α_5	α_6
1	1	1	1	-1					
2	2	$\frac{2}{3}$	1	$-\frac{4}{3}$	$\frac{1}{3}$				
3	3	$\frac{6}{11}$	1	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$			
4	4	$\frac{12}{25}$	1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$		
5	5	$\frac{60}{137}$	1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	
6	6	$\frac{60}{147}$	1	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$

The method is also called linear multi-step because, unlike the Runge-Kutta method, the expression of U is linear in ψ (the non-linearity resides inside ψ). All BDF methods are implicit methods. The first order BDF method is actually a one-step implicit method: the Backward Euler Method or the Implicit Euler Method. We can notice that, for the method to be self sufficient we need to know the solution for a number of steps. In the first order BDF method only the value of U at instant t^0 (the initial solution) is needed. For the scheme to be computed at $k-$order, k initial values $U^0, U^1, ..U^{k-1}$ are required to be $O(\Delta t^k)$ accurate. To obtain these values, an explicit method can be used. At each time step, a BDF method requires the solution of a non-linear system of equations. In order to solve the non-linear system of equations a Quasi- Newton method has been chosen at each time step.

Newton Method

In this work, we are interested in second order backward differentiation implicit scheme (see the Phd thesis of Ouvrard [84] for another application of the second order BDF scheme on the Navier-Stokes equations):

$$\alpha_{n+1} U^{(n+1)} + \alpha_n U^{(n)} + \alpha_{n-1} U^{(n-1)} + \Delta t^{(n)} \psi(U^{(n+1)}) = 0 \tag{1.18}$$

with the coefficients:

$$\alpha_{n+1} = \frac{1+2\tau}{1+\tau}, \alpha_n = -1-\tau, \alpha_{n-1} = \frac{\tau^2}{1+\tau},$$

and where $\Delta t^{(n)} = t^n + 1 - t^n$ and $\tau = \dfrac{\Delta t^{(n)}}{\Delta t^{(n-1)}}$, the ratio between the actual and the previous time step. A Newton process implies the definition of the iterate U^{n+1} using the previous iterate

U^n by linearising the system (1.18). To achieve the linearisation, we approximate the total flux at instant t^{n+1} through a Taylor formula:

$$\psi(U^{(n+1)}) = \psi(U^{(n)}) + \frac{d\psi}{dU}(U^{(n)})(U^{(n+1)} - U^{(n)}) + o(U^{(n+1)} - U^{(n)}).$$

Using the new expression of the total flux the BDF scheme (1.18) becomes:

$$\alpha_{n+1}U^{(n+1)} + \alpha_n U^{(n)} + \alpha_{n-1}U^{(n-1)} + \Delta t^{(n)}\psi(U^{(n)}) + \Delta t^{(n)}\frac{d\psi}{dU}(U^{(n)})(U^{(n+1)} - U^{(n)})) + \alpha_{n+1}U^{(n)} - \alpha_{n+1}U^{(n)} = 0,$$

and it can be rewritten under a linearised form for each node i as follows:

$$\begin{aligned}[\alpha_{n+1} + \Delta t^{(n)}\frac{d\psi_i}{dU_i}(U_i^{(n)})](U^{(n+1)} - U^{(n)}) = &- \Delta t^{(n)}\psi(U_i^{(n)}) - \alpha_{n+1}U_i^{(n)} \\ &- \alpha_n U_i^{(n)} - \alpha_{n-1}U_i^{(n-1)}.\end{aligned} \tag{1.19}$$

The Jacobian matrices $\frac{d\psi_i}{dU_i}(U_i^{(n)})$ are of size $N \times N$ and computed with a second order scheme for the convective flows. In order to reach a simple linear form we note:

$$R(U) = \psi(U_i^{(n)}) + \frac{\alpha_{n+1}}{\Delta t^{(n)}}U_i^{(n)} + \frac{\alpha_n}{\Delta t^{(n)}}U_i^{(n)} + \frac{\alpha_{n-1}}{\Delta t^{(n)}}U_i^{(n-1)},$$

the explicit residual and $J = \frac{d\psi_i}{dU_i}(U_i^{(n)})$, the Jacobian matrix. The linear system to solve becomes:

$$\left(\frac{\alpha_{n+1}}{\Delta t^{(n)}}I + J\right)\Delta U = -R(U). \tag{1.20}$$

The left hand side consists of the implicit operator and assembles the balance of the convective and diffusive fluxes.

Newton Algorithm:

- Let η be the relative tolerance for the residuum norm;
- Choose U^0, the initial solution;
- $i := 0$;
- Until convergence of Newton $\left(||R(U^{i+1})|| < \eta||U^0||\right)$:
  - solve $\left(\frac{\alpha_{n+1}}{\Delta t^{(n)}}I + J\right)\Delta U^i = -R(U^i) \Leftrightarrow$ call the Linear-Solver(J,R,ΔU^i),
  - update the iterate $U^{i+1} := U^i + \Delta U^i$,
  - $i = i + 1$;
- Update the solution $U = U^i$.

The Newton method needs to be combined with a fully linear solver, but the Jacobian computation is often inconvenient or expensive to compute and store. None of the proposed linear solvers (see below) requires the computation of Jacobian matrices. Only matrix vector products and are computed at each iteration. Moreover, to avoid costly computations at each iteration, each Jacobian matrix - vector product is approached by a finite difference scheme. In our simulations the convergence is reached only for a given tolerance that does not exceed 1e-6. The classical Newton method converges quadratically to the solution, but when approximative Jacobians and iterative techniques are used, an Inexact Newton method is constructed and

degrades the convergence. Asymptotic quadratic convergence is achievable [60] and it depends on the precision of the linear solver. During his PhD period, Remi Choquet [20] proves the local and linear convergence of the Inexact Newton method combined with the MFGMRES (FGMRES with restarts) linear solver.

Another remark is that the implicit schemes based on the lower-upper (LU) factorizations are stable and robust in any space dimension, even for high speed flows (Mach ≤ 20) or transonic flows (see [100]).

On approximative Jacobians

How do we choose the approximative expressions of the Jacobians? Based on what criteria? If many researchers have approached the discussion over the convective Jacobians (for example [93, 57]), there are little discussions in literature over the viscous Jacobians. To answer to these questions, we start by computing the exact viscous Jacobians. In the previous paragraph we have given a general formulation of the Newton Method applied to a BDF scheme. Let us proceed to an in-depth analysis of the linearisation of the system (1.18). We treat first the convective flux and second the diffusive fluxes. We rewrite the 2D system of Navier–Stokes equations already discretised in time, (1.18), as:

$$\begin{aligned}\alpha_{n+1}U^{(n+1)} + \alpha_n U^{(n)} + \alpha_{n-1}U^{(n-1)} + \Delta t^{(n)}\big[F^{n+1}_{Euler}(U^{n+1})\\ +G^{n+1}_{Euler}(U^{n+1}) + F^{n+1}_{vis}(U^{n+1},\nabla U^{n+1}) + G^{n+1}_{vis}(U^{n+1},\nabla U^{n+1})\big]\psi(U^{(n+1)}) = 0\end{aligned} \tag{1.21}$$

Let us explicit the linearisation steps for each component of the total flux.

$$\begin{aligned}
F^{n+1}_{Euler}(U^{n+1}) &= F^n_{Euler}(U^n) + \left(\frac{\partial F_{Euler}}{\partial U}\right)^n \Delta U^n + O((\Delta U^n)^2)\\
&= F^n_{Euler} + A\Delta U^n + O((\Delta U^n)^2),\\
G^{n+1}_{Euler}(U^{n+1}) &= G^n_{Euler}(U^n) + \left(\frac{\partial G_{Euler}}{\partial U}\right)^n \Delta U^n + O((\Delta U^n)^2)\\
&= G^n_{Euler} + B\Delta U^n + O((\Delta U^n)^2),\\
F^{n+1}_{vis}(U^{n+1},\nabla U^{n+1}) &= F^n_{vis}(U^n,\nabla U^n) + \left(\frac{\partial F_{vis}}{\partial U}\right)^n \Delta U^n + \left(\frac{\partial F_{vis}}{\partial U_x}\right)^n \Delta U^n_x\\
&\quad + \left(\frac{\partial F_{vis}}{\partial U_y}\right)^n \Delta U^n_y + O((\Delta U^n)^2,(\Delta U^n_x)^2,\Delta U^n_y)^2)\\
&= F^n_{vis} + A^v\Delta U^n + A^v_{\partial x}\Delta U^n_x + A^v_{\partial y}\Delta U^n_y\\
&\quad + O((\Delta U^n)^2,(\Delta U^n_x)^2,\Delta U^n_y)^2),\\
G^{n+1}_{vis}(U^{n+1},\nabla U^{n+1}) &= G^n_{vis}(U^n,\nabla U^n) + \left(\frac{\partial G_{vis}}{\partial U}\right)^n \Delta U^n + \left(\frac{\partial G_{vis}}{\partial U_x}\right)^n \Delta U^n_x\\
&\quad + \left(\frac{\partial G_{vis}}{\partial U_y}\right)^n \Delta U^n_y + O((\Delta U^n)^2,(\Delta U^n_x)^2,\Delta U^n_y)^2)\\
&= G^n_{vis} + B^v\Delta U^n + B^v_{\partial x}\Delta U^n_x + B^v_{\partial y}\Delta U^n_y\\
&\quad + O((\Delta U^n)^2,(\Delta U^n_x)^2,\Delta U^n_y)^2),
\end{aligned}$$

where we introduced the obvious notations:

$$
\begin{aligned}
F^n_{Euler} &= F^n_{Euler}(U^n), \qquad A = \left(\frac{\partial F_{Euler}}{\partial U}\right)^n,\\
G^n_{Euler} &= G^n_{Euler}(U^n), \qquad B = \left(\frac{\partial G_{Euler}}{\partial U}\right)^n,\\
F^n_{vis} &= F^n_{vis}(U^n, \nabla U^n), A^v = \left(\frac{\partial F_{vis}}{\partial U}\right)^n, A^v_{\partial x} = \left(\frac{\partial F_{vis}}{\partial U_x}\right)^n, A^v_{\partial y} = \left(\frac{\partial F_{vis}}{\partial U_y}\right)^n,\\
G^n_{vis} &= G^n_{vis}(U^n, \nabla U^n), B^v = \left(\frac{\partial G_{vis}}{\partial U}\right)^n, B^v_{\partial x} = \left(\frac{\partial G_{vis}}{\partial U_x}\right)^n, B^v_{\partial y} = \left(\frac{\partial G_{vis}}{\partial U_y}\right)^n.
\end{aligned}
$$

Or, the terms in $O((\Delta U^n)^2) = O\left(\left(\Delta t \frac{\partial U}{\partial t}\right)^2\right) = O(\Delta t^2)$ with $\Delta t = t^{n+1} - t^n$ can be neglected for a Newton type linearisation. With these new notations, the system to solve can be rewritten as in (1.22) where the unknowns are not only the conservative vector U at each time step, but their derivatives as well:

$$
\begin{aligned}
\alpha_{n+1}\Delta U^n + \Delta t \frac{\partial}{\partial x}\big(A\Delta U^n + A^v\Delta U^n + A^v_{\partial x}\Delta U^n_x + A^v_{\partial y}\Delta U^n_y\big)\\
+\Delta t \frac{\partial}{\partial y}\big(B\Delta U^n + B^v\Delta U^n + B^v_{\partial x}\Delta U^n_x + B^v_{\partial y}\Delta U^n_y\big) = -\Delta t R^n,
\end{aligned}
\tag{1.22}
$$

where $R^n = F^n_{Euler} + G^n_{Euler} + F^n_{vis} + G^n_{vis} + \frac{\alpha_n}{\Delta t}U^{(n)} + \frac{\alpha_{n-1}}{\Delta t}U^{(n-1)}$.

The next step is the calculation of the exact expression of $A, A^v, A^v_{\partial x}, A^v_{\partial y}, B, B^v, B^v_{\partial x}$ and $B^v_{\partial y}$. They are identified from the computation of the differentials of the viscous fluxes, $F_{vis} + G_{vis}$. A detailed calculation of the differentials of viscous fluxes can be found in appendix A.

Once the expressions of the exact Jacobians have been found, we can discuss over the choice of their approximations. We base our study on the similar reasoning of computing the Jacobians, made by Beam and Warming [4] and further studied by Coakley [21, 22]. They propose two simplifications in the expression of (1.22). We translate and adapt their algorithm steps to our system of equations. First, they propose an equivalent expression of the Taylor approximations for the viscous fluxes:

$$
\begin{aligned}
F^{n+1}_{vis} &= F^n_{vis} + A^v\Delta U^n + A^v_{\partial x}\Delta U^n_x + A^v_{\partial y}\Delta U^n_y + O((\Delta t)^2),\\
&= F^n_{vis} + \big(A^v + \frac{\partial}{\partial x}A^v_{\partial x}\big)\Delta U^n + \frac{\partial}{\partial x}\big(A^v_{\partial y}\Delta U^n_y\big) + A^v_{\partial y}\Delta U^n_y + O((\Delta t)^2)\\
G^{n+1}_{vis} &= G^n_{vis} + B^v\Delta U^n + B^v_{\partial x}\Delta U^n_x + B^v_{\partial y}\Delta U^n_y + O((\Delta t)^2),\\
&= G^n_{vis} + B^v_{\partial x}\Delta U^n_x + \big(B^v + \frac{\partial}{\partial y}B^v_{\partial y}\big)\Delta U^n + \frac{\partial}{\partial y}\big(B^v_{\partial y}\Delta U^n_y\big) + O((\Delta t)^2).
\end{aligned}
$$

In what concerns the cross-derivative terms, Beam and Warming said about this kind of treatment that *encounter considerable difficulty in constructing efficient factored algorithm.* Based on the finite differences discretisation of a second order derivative $O(\Delta t^2) = (\Delta t)^2\frac{\partial^2 F_{vis}(U^n_y)}{\partial t} = F_{vis}(U^{n+1}_y) - 2F_{vis}(U^n_y) + F_{vis}(U^{n-1}_y)$ and $O(\Delta t^2) = (\Delta t)^2\frac{\partial^2 G_{vis}(U^n_x)}{\partial t} = G_{vis}(U^{n+1}_x) - 2G_{vis}(U^n_x) + G_{vis}(U^{n-1}_x)$, the cross derivatives can be calculated without loss accuracy from the previous known values, $F_{vis}(U^{n+1}_x) - F_{vis}(U^n_x) = F_{vis}(U^n_x) - F_{vis}(U^{n-1}_x) + O(\Delta t^2)$, $G_{vis}(U^{n+1}_x) - G_{vis}(U^n_x) = G_{vis}(U^n_x) - G_{vis}(U^{n-1}_x) + O(\Delta t^2)$, it can even be advantageous to neglect them, so to neglect the terms $A^v_{\partial y}\Delta U^n_y$ and $B^v_{\partial x}\Delta U^n_x$ for some calculations. Moreover, in their reasoning the expression $\frac{\partial}{\partial x}\big((A + A^v + \frac{\partial}{\partial x}A^v_{\partial x})\Delta U^n\big)$ denotes $\frac{\partial}{\partial x}\big(A + A^v + \frac{\partial}{\partial x}A^v_{\partial x}\big)\Delta U^n$, meaning that all second order spatial derivatives are also neglected.

Under its original form, the system (1.22) is not well posed since the number of unknowns exceeds the number of equations. We use second order Finite Volume scheme to discretise the Navier–Stokes system of equations. Assuming, locally, that the cross-derivatives and the second order spatial derivatives are zero should not decrease the accuracy. Moreover, all the assumptions of Beam and Warming are adopted in elsA and have proved their efficiency in solving large systems of data in industrial codes. These approximative Jacobians are easier to compute and will be preferred in our simulations. Let us give their expressions for modelling the convective and diffusive fluxes:

$$A = \begin{pmatrix} 0 & \frac{S}{L} & 0 & 0 \\ \big(-e(\gamma-1)-u^2\big)\frac{S}{L} & \big(u(1-\gamma)\big)\frac{S}{L} & (1-\gamma)v\frac{S}{L} & (\gamma-1)\frac{S}{L} \\ -uv\frac{S}{L} & v\Delta y & u\frac{S}{L} & 0 \\ (h-e)(1-\gamma)u\frac{S}{L} & (1-\gamma)(u^2-h)\frac{S}{L} & (1-\gamma)uv\frac{S}{L} & \gamma u\frac{S}{L} \end{pmatrix},$$

$$B = \begin{pmatrix} 0 & 0 & \frac{S}{L} & 0 \\ -uv\frac{S}{L} & v\frac{S}{L}+\rho_B & u\frac{S}{L} & 0 \\ (e(\gamma-1)-v^2)\frac{S}{L} & (1-\gamma)u\frac{S}{L} & (1-\gamma)v\frac{S}{L} & (\gamma-1)\frac{S}{L} \\ (h-z)(1-\gamma)v\frac{S}{L} & (1-\gamma)uv\frac{S}{L} & (1-\gamma)(v^2-h)\frac{S}{L} & \gamma v\frac{S}{L} \end{pmatrix},$$

$$A^v + \frac{\partial}{\partial x}A^v_{\partial x} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -\frac{4}{3}\frac{S}{L}\frac{\mu}{\rho}u & \frac{4}{3}\frac{S}{L}\frac{\mu}{\rho} & 0 & 0 \\ -\frac{S}{L}\frac{\mu}{\rho}v & 0 & \frac{S}{L}\frac{\mu}{\rho} & 0 \\ -\big[\frac{4}{3}u^2+v^2+\frac{\gamma}{Pr}(E-2e)\big]\frac{S}{L}\frac{\mu}{\rho} & \big[\frac{4}{3}-\frac{\gamma}{Pr}\big]\frac{S}{L}\frac{\mu}{\rho}u & \big[1-\frac{\gamma}{Pr}\big]\frac{S}{L}\frac{\mu}{\rho}v & \frac{\gamma}{Pr}\frac{S}{L}\frac{\mu}{\rho} \end{pmatrix},$$

$$B^v + \frac{\partial}{\partial y}B^v_{\partial y} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -\frac{S}{L}\frac{\mu}{\rho}u & \frac{S}{L}\frac{\mu}{\rho} & 0 & 0 \\ -\frac{4}{3}\frac{S}{L}\frac{\mu}{\rho}v & 0 & \frac{4}{3}\frac{S}{L}\frac{\mu}{\rho} & 0 \\ -\big[u+\frac{4}{3}v+\frac{\gamma}{Pr}(E-2e)\big]\frac{S}{L}\frac{\mu}{\rho} & \big[1-\frac{\gamma}{Pr}\big]\frac{S}{L}\frac{\mu}{\rho}u & \big[\frac{4}{3}-\frac{\gamma}{Pr}\big]\frac{S}{L}\frac{\mu}{\rho}v & \frac{\gamma}{Pr}\frac{S}{L}\frac{\mu}{\rho} \end{pmatrix},$$

where S denotes the surface of one cell (it has the same definition for a 3D computation), and L the length of one cell in the chosen direction. For the x-direction we get $\dfrac{S}{L} = \dfrac{\Delta x \Delta y}{\Delta x} = \Delta y$. Excepted S and L, all symbols presented in the Jacobian definition have been already defined in chapter II and can also be find in the nomenclature of this work. We note that we found (see appendix A) exactly the same expressions of $A^v_{\partial x}$ and $B^v_{\partial y}$.

Another possible Jacobian approximation is to replace the expression of viscous Jacobians with a diagonal expression of its spectral radius. It can be seen as a smoother for solving stiff problems.

Linear resolution

The construction of implicit schemes implies the solution of large linear systems of equations for each iteration. The linear system is usually sparse or multi-diagonal by blocks. The resulting linear system (1.20) can be written under the general form $Ax = b$, where $x = \Delta U$ it the increment of the solution, $b = -R(U)$ is the explicit residual and $A = \Big(\dfrac{\alpha_{n+1}}{\Delta t^{(n)}}I + J\Big)$.

Since the size of the system is very large, and solving the system is the most costly part of the computation, it is usually solved in an approximate way adapted to the form of the matrix. In this thesis, three linear solvers have been studied: GMRES, Gauss Seidel and the Relaxation method. We present briefly these three methods.

GMRES Method

The GMRES method or the generalised minimal residual method is a projection method based on the projection of the approximated solution on K_n orthogonal to $L = AK_n$, where K_n is the n-th sub-space of Krylov with $v_1 = r_0/||r_0||$. It searches an approximated solution $x_n \in K_n$ of the exact solution of $Ax = b$ that minimises the residual norm $||Ax_n - b||$. The approximated solution by GMRES is the unique vector $x_0 + K_n$ that minimises $J(x) = ||b - Ax||_2$. Like other iterative methods, GMRES is often combined with accelerators to increase the speed of convergence. In this work a flexible right preconditioned GMRES algorithm has been chosen to be analysed. The main idea is to solve:

$$AM^{-1}u = b \quad u = Mx,$$

where M is the preconditioner, a positive symmetrical matrix that approximates A. The difference between this algorithm and the right preconditioned GMRES algorithm is that the right preconditioner can change at each step.

FGMRES algorithm:

- Define the initial residual $r_0 = b - Ax_0 \quad \beta = ||r_0||_2$ and $v_1 = r_0\beta$;
- for j from 1 to n do:
  - $z_j := M_j^{-1} v_j$,
  - $w := Az_j$,
  - for i from 1 to j do:
    - $h_{i,j} := (w, v_i)$,
    - $w := w - h_{i,j} v_j$,
  - $h_{j+1,j} = ||w||_2$ *et* $v_{j+1} = w/h_{j+1,j}$;
  - $Z_n := [z_1, .., z_n]$. $H_m = \{h_{i,j}\}_{1 \leq i \leq j+1; 1 \leq j \leq m}$;
- $y_n = \text{argmin}_y ||\beta e_1 - H_m y||_2$, $x_n = x_0 + Z_n y_n$;
- if y_n satisfies the stopping criteria exit the iteration;
- else $x_0 = x_n$ and the next iteration can begin.

The FGMRES algorithm is numerically sensitive and more costly than the Gauss-Seidel or the relaxation ones. It can also lead to the stagnation of the residual norm. Because of these reasons the Gauss-Seidel method and the Relaxation one have been preferred for the scalability study.

Gauss-Seidel Method

This iterative method was developed to solve large sparse linear systems of equations. The system to solve is:

$$Ax = b$$

We suppose that the invertible matrix A can be decomposed as a sum of three matrices: $\mathcal{L}$ the upper triangular part of A, $\mathcal{D}$ its diagonal part and $\mathcal{U}$ the lower triangular part of A:

$$A = \mathcal{L} + \mathcal{D} + \mathcal{U}.$$

The system to solve has the following form:

$$(\mathcal{L}+\mathcal{D}+\mathcal{U})x = b. \tag{1.23}$$

Let us note $M = \mathcal{L}+\mathcal{D}$. Since $\mathcal{D}$ contains, by hypothesis, only non zeros elements on the diagonal, $\det(M) \neq 0$ so M is invertible. Its inverse is easy to build since M is defined as a lower triangular matrix. We obtain an iterative method called the Gauss-Seidel method per points that we can write under the notation:

$$(\mathcal{L}+\mathcal{D})x_{k+1} = -\mathcal{U}x_k + b,\ k \in \mathbb{N}$$

and more precisely

$$x_{k+1} = -(\mathcal{L}+\mathcal{D})^{-1}\mathcal{U}x_k + (\mathcal{L}+\mathcal{D})^{-1}b.$$

The matrix $(\mathcal{L}+\mathcal{D})^{-1}\mathcal{U}$ is called the Gauss-Seidel matrix per points. One advantage of this method is that it only needs one vector to store. As we shall see later, the Gauss-Seidel algorithm is equivalent to an alternating Schwarz domain decomposition algorithm.

Relaxation Method

We keep the same decomposition of A $(= \mathcal{L}+\mathcal{D}+\mathcal{U})$ as for the Gauss-Seidel method. The relaxation method consists in computing a series of approximated solutions of the exact system:

$$(\mathcal{L}+\mathcal{D}+\mathcal{U})x = b.$$

We consider a relaxation method with forward and backward sweeps through the domain:

$$\begin{array}{rl}(\mathcal{L}+\mathcal{D})x^{p+\frac{1}{2}} & = b - \mathcal{U}x^{p} \\ (\mathcal{D}+\mathcal{U})x^{p+1} & = b - \mathcal{L}x^{p+\frac{1}{2}}\end{array},$$

where p is the number of relaxation cycles. The speed of convergence depends on the choice of the number of relaxation cycles. Usually 2 cycles is a good choice. For a null initial solution we have:

$$\begin{array}{rl}x^{0} & = 0 \\ (\mathcal{L}+\mathcal{D})x^{\frac{1}{2}} & = b \\ (\mathcal{D}+\mathcal{U})x^{1} & = b - \mathcal{L}x^{\frac{1}{2}}\end{array},$$

thus

$$\begin{array}{rl}(\mathcal{D}+\mathcal{U})x^{1} & = \mathcal{D}x^{\frac{1}{2}} \\ (\mathcal{L}+\mathcal{D})D^{-1}(\mathcal{D}+\mathcal{U})x^{1} & = b.\end{array}$$

One cycle of the relaxation method with a null initialisation is equivalent to an approximate LDU method.

Application to the 3D Navier–Stokes Equations

Let us recall the linearised system to solve, (1.20):

$$\left(\frac{\alpha_{n+1}}{\Delta t^{(n)}}I + J\right)\Delta U = -R(U).$$

Let A be the Jacobian matrix of the convective or Euler flux in the $x-$direction $F(U)$ and B the Jacobian matrix of the convective in the $y-$direction $G(u)$, and C the Jacobian matrix of the convective flow in the $z-$direction $H(u)$, using the conservative variables. We define the approximated positive and negative parts of these Jacobian as follows:

$$A^{\pm} = \frac{1}{2}(A \pm |A|),\quad B^{\pm} = \frac{1}{2}(B \pm |B|),\quad C^{\pm} = \frac{1}{2}(C \pm |C|),$$

where $E \in \{A, B, C\}$ is diagonalisable and $|E| = M|\Lambda|M^{-1}$, M is an invertible matrix and Λ is a diagonal matrix.

The Jacobian parts are constructed in such a way that the eigenvalues of the A^+, B^+ and C^+ matrices are non-negative and those of A^-, B^- and C^- matrices are non-positive. Let $A^v(U)$ be the Jacobian matrix of the viscous flow in the $x-$direction $F^v(U)$, $B^v(U)$ the Jacobian matrix of the viscous flow in the $y-$direction $G^v(U)$, and $C^v(U)$ the Jacobian matrix of the viscous flow in the $z-$direction $H^v(U)$ using the conservative variables.

For the last part of this sections and for simplification purpose, we denote the index of the 3D cell $\Omega_{i,j,k}$ by only Ω. $\Omega - 2x$ will denote $\Omega_{i-2,j,k}$ and similarly $\Omega - 2y$ will denote $\Omega_{i,j-2,k}$ and $\Omega - 2z$ will denote $\Omega_{i,j,k-2}$. With these previous notations we rewrite the linear system to solve as:

$$\Big(\frac{\alpha_{n+1}}{\Delta t^{(n)}} I + A_\Omega + B_\Omega + C_\Omega + A^v_\Omega + B^v_\Omega + C^v_\Omega\Big)\,\Delta U_\Omega = -R_\Omega. \tag{1.24}$$

We replace these Jacobians with the sum of their positive and negative parts. We write the convective part by the following relation (upwind: $\dfrac{\partial^+}{\partial x}$ and downwind: $\dfrac{\partial^-}{\partial x}$):

$$\begin{aligned}\frac{\partial^-}{\partial x}A^+_\Omega\Delta U_\Omega + \frac{\partial^+}{\partial x}A^-_\Omega\Delta U_\Omega &= (A^+_\Omega\Delta U_\Omega - A^+_{\Omega-x}\Delta U_{\Omega-x})\\ &\quad +(A^-_{\Omega+x}\Delta U_{\Omega+x} - A^-_\Omega\Delta U_\Omega),\end{aligned}$$

and similarly in the other directions:

$$\begin{aligned}\frac{\partial^-}{\partial y}B^+_\Omega\Delta U_\Omega + \frac{\partial^+}{\partial y}B^-_\Omega\Delta U_\Omega &= (B^+_\Omega\Delta U_\Omega - B^+_{\Omega-y}\Delta U_{\Omega-y})\\ &\quad +(B^-_{\Omega+y}\Delta U_{\Omega+y} - B^-_\Omega\Delta U_\Omega),\end{aligned}$$

$$\begin{aligned}\frac{\partial^-}{\partial z}C^+_\Omega\Delta U_\Omega + \frac{\partial^+}{\partial z}C^-_\Omega\Delta U_\Omega &= (C^+_\Omega\Delta U_\Omega - C^+_{\Omega-z}\Delta U_{\Omega-z}),\\ &\quad +(C^-_{\Omega+z}\Delta U_{\Omega+z} - C^-_\Omega\Delta U_\Omega).\end{aligned}$$

The diffusive balance is treated in its original form, the approximate linearisation is evaluated at cell centres as follows:

$$\begin{aligned}\frac{\partial}{\partial x}A^v_\Omega\Delta U_\Omega &= \tfrac{1}{4}\big(A^v_{\Omega-2x}\Delta U_{\Omega-2x} - 2A^v_\Omega\Delta U_\Omega + A^v_{\Omega+2x}\Delta U_{\Omega+2x}\big),\\ \frac{\partial}{\partial y}B^v_\Omega\Delta U_\Omega &= \tfrac{1}{4}\big(B^v_{\Omega-2y}\Delta U_{\Omega-2y} - 2B^v_\Omega\Delta U_\Omega + B^v_{\Omega+2y}\Delta U_{\Omega+2y}\big),\\ \frac{\partial}{\partial z}C^v_\Omega\Delta U_\Omega &= \tfrac{1}{4}\big(C^v_{\Omega-2z}\Delta U_{\Omega-2z} - 2C^v_\Omega\Delta U_\Omega + C^v_{\Omega+2z}\Delta U_{\Omega+2z}\big).\end{aligned}$$

We then rewrite the system to solve:

$$\begin{aligned}&\frac{\Delta t}{4}(A^v_{\Omega-2x}\Delta U_{\Omega-2x} + B^v_{\Omega-2y}\Delta U_{\Omega-2y} + C^v_{\Omega-2z}\Delta U_{\Omega-2z})\\ &\quad +\Delta t(-A^+_{\Omega-x}\Delta U_{\Omega-x} - B^+_{\Omega-y}\Delta U_{\Omega-y} - C^+_{\Omega-z}\Delta U_{\Omega-z})\\ &+(\alpha_{n+1}I +\Delta t(A^+_\Omega - A^-_\Omega + B^+_\Omega - B^-_\Omega + C^+_\Omega - C^-_\Omega - \frac{1}{2}A^v_\Omega - \frac{1}{2}B^v_\Omega - \frac{1}{2}C^v_\Omega))\Delta U_\Omega\\ &\quad +\Delta t(A^-_{\Omega+x}\Delta U_{\Omega+x} + B^-_{\Omega+y}\Delta U_{\Omega+y} + C^-_{\Omega+z}\Delta U_{\Omega+z})\\ &\quad +\frac{\Delta t}{4}(A^v_{\Omega+2x}\Delta U_{\Omega+2x} + B^v_{\Omega+2y}\Delta U_{\Omega+2y} + C^v_{\Omega+2z}\Delta U_{\Omega+2z}) = -\Delta t R_\Omega.\end{aligned} \tag{1.25}$$

We extract the diagonal part, D_Ω, and approximate the values of $|A|, |B|$ and $|C|$ with the spectral radius of each one of the Jacobian matrices. This approximation was proposed by Yoon and Jameson [100] when they propose a new multigrid relaxation scheme, lower-upper

symmetric successive over-relaxation ($LU-SSOR$) scheme to solve the Euler and Navier–Stokes equations.

$$\begin{aligned}\mathcal{D}_\Omega &= (\alpha_{n+1}I + \Delta t(A^+_\Omega - A^-_\Omega + B^+_\Omega - B^-_\Omega + C^+_\Omega - C^-_\Omega + \frac{1}{2}A^v_\Omega + \frac{1}{2}B^v_\Omega + \frac{1}{2}C^v_\Omega))\\ &= (\alpha_{n+1}I + \Delta t(|A_\Omega| + |B_\Omega| + |C_\Omega| + \frac{1}{2}A^v_\Omega + \frac{1}{2}B^v_\Omega + \frac{1}{2}C^v_\Omega))\\ &\approx (\alpha_{n+1} + \Delta t(\rho_A + \rho_B + \rho_C + \frac{1}{2}\rho_{A^v} + \frac{1}{2}\rho_{B^v} + \frac{1}{2}\rho_{C^v}))I\end{aligned}$$

The spectral radius, ρ_*, is the maximum eigenvalue of a Jacobian matrix defined as:

$$\rho_A = \max(u-a, u+a), \rho_B = \max(v-a, v+a), \rho_C = \max(w-a, w+a),$$

$$\rho_{A^v} = \max(\frac{4}{3}\mu, \gamma\frac{\mu}{Pr}), \rho_{B^v} = \max(\frac{4}{3}\mu, \gamma\frac{\mu}{Pr}), \rho_{C^v} = \max(\frac{4}{3}\mu, \gamma\frac{\mu}{Pr}).$$

We make the following notations:

$$\mathfrak{L} = \mathcal{L} + \mathcal{D}, \quad \mathfrak{D} = \mathcal{D}, \quad \mathfrak{U} = \mathcal{U} + \mathcal{D},$$

and resume the problem to a $\mathfrak{L}\mathfrak{D}^{-1}\mathfrak{U}$ factorisation:

$$\mathfrak{L}\mathfrak{D}^{-1}\mathfrak{U}\Delta U^n = -R,$$

that we can solve within three steps:

1. Solve the $\mathfrak{L}$-Relaxation
$$\mathfrak{L}\Delta U^{\frac{1}{3}} = -R,$$
where $\mathfrak{L} = \mathcal{L} + \mathcal{D}$ is a lower triangular matrix.
The iterations are:
$$\Delta U^{\frac{1}{3},0}_\Omega = -\mathfrak{D}^{-1}R^0_\Omega,$$
$$\Delta U^{\frac{1}{3},1}_\Omega = -\mathfrak{D}^{-1}R^1_\Omega + \mathfrak{D}^{-1}(A^+_{\Omega-x}\Delta U^{\frac{1}{3},0}_{\Omega-x} + B^+_{\Omega-y}\Delta U^{\frac{1}{3},0}_{\Omega-y} + C^+_{\Omega-z}\Delta U^{\frac{1}{3},0}_{\Omega-z}),$$
for instant $k, 2 \le k \le n$:
$$\begin{aligned}\Delta U^{\frac{1}{3},k}_\Omega = -\mathfrak{D}^{-1}R^k_\Omega + \mathfrak{D}^{-1}[&\frac{1}{4}(A^v +_{\Omega-2x} \Delta U^{\frac{1}{3},k-1}_{\Omega-2x} + B^v_{\Omega-2y}\Delta U^{\frac{1}{3},k-1}_{\Omega-2y} + C^v_{\Omega-2z}\Delta U^{\frac{1}{3},k-1}_{\Omega-2z})\\ &+ A^+_{\Omega-x}\Delta U^{\frac{1}{3},k-1}_{\Omega-x} + B^+_{\Omega-y}\Delta U^{\frac{1}{3},k-1}_{\Omega-y} + C^+_{\Omega-z}\Delta U^{\frac{1}{3},k-1}_{\Omega-z}].\end{aligned}$$

2. Solve the diagonal
$$\mathfrak{D}^{-1}\Delta U^{\frac{2}{3}} = \Delta U^{\frac{1}{3}},$$
which is equivalent to:
$$\Delta U^{\frac{2}{3}} = \mathfrak{D}\Delta U^{\frac{1}{3}}.$$

3. Solve the $\mathfrak{U}$-relaxation
$$\mathfrak{U}\Delta U^n = \Delta U^{\frac{2}{3}},$$
where $\mathfrak{U} = \mathcal{U} + \mathcal{D}$ is an upper triangular matrix. The iterative resolution starts with the last terms:
$$\begin{aligned}\Delta U^n_\Omega &= -\mathfrak{D}^{-1}R^n_\Omega,\\ \Delta U^{n-1}_\Omega &= -\mathfrak{D}^{-1}R^{n-1}_\Omega - \mathfrak{D}^{-1}(A^-_{\Omega+x}\Delta U^{\frac{2}{3},n}_{\Omega+x} + B^-_{\Omega+y}\Delta U^{\frac{2}{3},n}_{\Omega+y} + C^-_{\Omega+z}\Delta U^{\frac{2}{3},n}_{\Omega+z}),\end{aligned}$$
for $k < n-1$ we compute:
$$\begin{aligned}\Delta U^{k-1}_\Omega = &-\mathfrak{D}^{-1}R^{k-1}_\Omega - \mathfrak{D}^{-1}[A^-_{\Omega+x}\Delta U^{\frac{2}{3},k}_{\Omega+x} + B^-_{\Omega+y}\Delta U^{\frac{2}{3},k}_{\Omega+y} + C^-_{\Omega+z}\Delta U^{\frac{2}{3},k}_{\Omega+z}\\ &+\tfrac{1}{4}(A^v_{\Omega+2x}\Delta U^{\frac{2}{3},k}_{\Omega+2x} + B^v_{\Omega+2y}\Delta U^{\frac{2}{3},k}_{\Omega+2y} + C^v_{\Omega+2z}\Delta U^{\frac{2}{3},k}_{\Omega+2z})].\end{aligned}$$

The steps 1 and 3 can be solved directly by forward and backward substitution due to the triangular form of matrices $\mathcal{L}+\mathcal{D}$ and $\mathcal{U}+\mathcal{D}$.

Chapter 2

Schwarz based Domain Decomposition Methods (DDMs)

Contents

Parallel computation has opened a wide horizon for new precise and complex simulations in a range of applications including traffic flow, weather predictions, seismic structures, astrophysics and medicine. The main idea is to split large problems that model physical phenomena into smaller sub-problems that can be treated in parallel. The splitting techniques are called domain decomposition methods (DDMs), they are based on the Divide and Conquer principle and can be seen as a parallelisation tool. DDM permits to simulate more and more complicated physical phenomena modelled by partial differential equations as it allows the computation of a larger size problem. For simulations that are already running it accelerates the process and gives a solution faster.

The efficiency of a parallel computation depends directly on the chosen domain decomposition method. An efficient DDM needs to respect several factors: the initial problem on a complicated geometry has to be split into sub-problems on simpler geometries; the resulting algorithm has to be easy to implement and to have a good parallel performance; the parallel solution must equal the global solution. The necessity to fulfill of these three factors leads to the elaboration of a large spectrum of domain decomposition algorithms that are usually focused on the type of the problem, linear or non-linear, hyperbolic, parabolic or elliptic. For linear systems, DDM can be seen as a preconditioner for Krylov subspace accelerator techniques. For non-linear cases they can be seen as a preconditioner of the solution of the linear system. Moreover, DDM allows modelling and coupling different physics in different sub-domains, for example modelling the ocean coupled with the atmosphere [33, 7]. They allow coupling different geometries, different discretisations, like Cartesian modelling with triangulation [8, 55]. We can imagine different fluid behaviour described by the Navier–Stokes equations coupled with the Euler equations [8, 55].

In the continuous attempt to elaborate more efficient parallel algorithms to compute the numerical solution of a system of equations, three different types of parallelism can be identified ([11, 88]).

- Parallelism of the method [71], or methods that compute blocks of values simultaneously. Two representative examples are the Shur complement method based on parallel local factorisation of sub-blocks of matrices and the FETI (Finite Element Tearing and Interconnecting) method, an iterative substructuring method for solving linear system of equations.
- Parallelism over the space or Schwarz based domain decompositions [87, 64] (described below).
- Parallelism over the time [3, 63] also known as the parareal algorithm was introduced in 2001 by J.-L. Lions and al [63]. It is based on a predictor-corrector algorithm which uses a sequentially computed coarse grid in time to predict (and then to correct) solutions. They become initial solutions for fine grids in time sequentially solved inside one sub-domain, but with parallel implementation of different sub-domains.

An efficient method could assemble elements from the three types. Usually, only space domain decomposition method is used to provide high-performing algorithms in many fields of numerical applications. In this study we will only consider the overlapping domains (fig.2.1, left) decomposition method that includes Schwarz based decomposition and not the non-overlapping domain decomposition (fig.2.1, right), referred in literature as sub-structuring or Schur complement methods.

In this chapter we will first overview the classical Schwarz based domain decomposition methods, that are space domain decomposition and, secondly the Schwarz Waveform Relaxation (SWR) domain decomposition method based on [62] and developed in [40], a method that includes the time in its decomposition. We will distinguish different domain decomposition methods applied to explicit and implicit discretisations.

We state the main task of this work: the analysis and the improvement of SWR domain

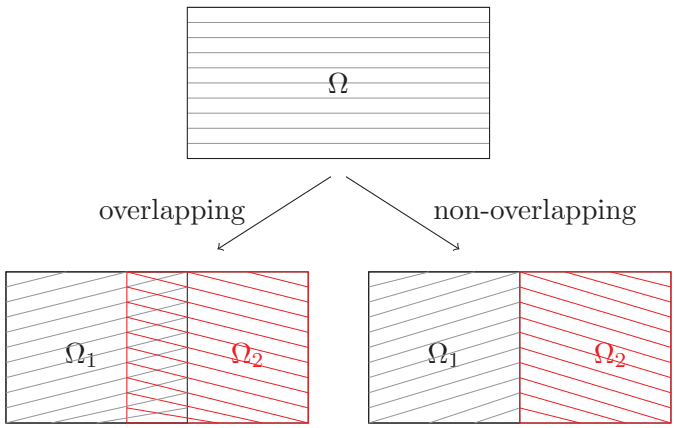


Figure 2.1: Overlapping (left) and non-overlapping (right) decomposition of a domain Ω.

decomposition method. We propose and discuss a new way to improve the SWR domain decomposition method.

2.1 Classical Schwarz domain decomposition methods

Many engineering problems can be formulated as boundary problems on complex geometries. On their original form, these problems are difficult or costly to solve and they are not particularly adapted to parallel computing.

In 1869, H.A. Schwarz [87] proposes a first domain decomposition for a global computation on a complex geometry into an iterative one on simple geometries to solve the Laplace equation. The goal of this decomposition is to reduce the problem on an unknown geometry into problems on simple geometries, possible to solve. Schwarz splits the global problem (fig.2.2, left) into overlapping sub-problems and solves alternately each one of them by exchanging interface conditions, thus the name of multiplicative or alternating Schwarz method. It was also the first time that one managed to prove that the Laplace equation on a bounded domain with Dirichlet boundary conditions admits a unique solution on arbitrary domains. Proofs already existed for a rectangular domain (Fourier, 1807) or a circular domain (Poisson, 1815). Schwarz proved the convergence of his method using the maximum principle, first for a composed domain of a disk and a rectangle, then he extended it, adding other disks or rectangles to the initial domain, obtaining more difficult configurations. He also proposed a physical device to prove his method: a vacuum pump with two cylinders (fig.2.2, right). The alternative pump of the cylinders produces a vacuum in the inner chamber. A more detailed background on the origins of the alternating Schwarz method has recently been published by Gander and Wanner in a conference article [47]. A general background on all Schwarz Methods can be found in [39] and on overlapping methods in [13].

A century later, P.-L. Lions re-explores Schwarz idea and gives a parallel formulation for the purpose of parallel computing, which is the basis of many domain decomposition methods. He publishes a series of three papers titled "On the Schwarz alternating method" [64, 65, 66] where he conducts a mathematical investigation of the convergence properties of the Schwarz alternating method for different linear and non-linear problems. In his first paper he proves the convergence of the Schwarz alternating method for linear problems: the Laplace equation and the Stokes equations, and non-linear monotone problems such as the Heat problem. He shows that the Schwarz method to solve the Laplace problem is equivalent to a sequence of projections in a Hilbert space and gives a completely different convergence proof using projections. Lions combines the original method with a time discretisation, extends the method from two to more

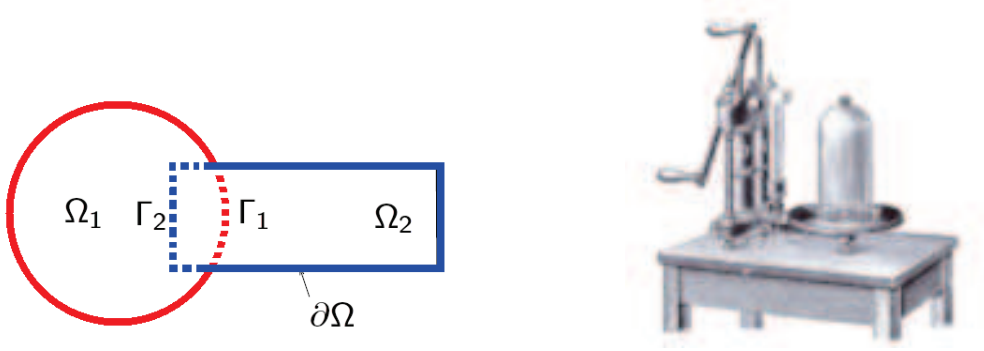


Figure 2.2: Schwarz first decomposition (left). Two level vacuum pump used to physically interpret the alternating Schwarz method (right)

sub-domains and proposes several variants of the Schwarz method, among them the parallel method. In the second article, [65], he continues the study giving convergence results of the alternating method for uniformly elliptic equations, degenerate elliptic equations and parabolic equations with overlapping and non-overlapping domains. The convergence proof is conducted via the maximum principle or pure probabilistic using a stochastic interpretation. In the last part of the article he offers some hint to optimise the domain distribution in order to accelerate the convergence. In the third article, [66], Lions observes that applications can be severely restricted by the necessity of overlapping sub-domains, and the computation is more expensive. He then proposes a new non-overlapping algorithm by changing the Dirichlet communication conditions into Robin communication conditions between two sub-domains. He uses the energy estimates to prove that the new algorithm is convergent for an arbitrary number of sub-domains. P.L. Lions papers can be seen as a consolidation of the properties of Schwarz algorithms, which are considered robust and convergent indifferent to the type of equations considered.

We begin our analysis by solving the Cauchy problem associated to the 1D Navier–Stokes system of equations and Dirichlet type boundary conditions:

$$\text{Find U that satisfies} \begin{cases} \partial_t U + \partial_x F_{Euler}(U) + \partial_x F_{vis}(U) = 0 & \text{in } \Omega \times [0,T], \\ U = U_0 & \text{on } \Omega \times \{0\}, \\ U = g & \text{on } \partial\Omega \times [0,T], \end{cases} \tag{2.1}$$

where U is the unknown vector of conservative variable $U = (\rho, \rho u, \rho E)$, $U_0 = (\rho_0, (\rho u)_0, (\rho E)_0)$ is the initial solution, $\Omega = [a,b]$ is the 1D domain, and $[0,T]$ the duration of the simulation. Let us split the original domain Ω into two overlapping sub-domains Ω_1, and Ω_2, with interface Γ and denote $\Gamma_1 = \partial\Omega_1 \cap \Omega_2$ and $\Gamma_2 = \partial\Omega_2 \cap \Omega_1$ respectively the boundaries of Ω_1 and Ω_2 that intersect with the opposed sub-domain. We divide the time axis in N intervals Δt: $0 = t^0 < t^1 < .. < t^{N-1} < t^N = T$. We are searching an approximation U^k of U at instant t^k, $1 \leq k \leq N$. The chosen meshes are of two kinds: coincident and non-coincident. When coincident meshes are used, the ghost cells receive their values from the internal cells of the neighbour sub-domain. When the meshes are non-coincident, the values of the ghost cells are found by quadratic interpolation within the neighbour internal cells.

2.1.1 Alternating Schwarz Algorithm

The main idea is to use a fixed point algorithm where, at each iteration, the boundary conditions on Γ_1 and Γ_2 are updated. Their values are the traces of the opposed domain from the previous iteration. The result is a two steps algorithm:

1. Choose the initial solution for each sub-domain;

2. For $k = 1, 2, ...$ solve

$$\begin{cases} \partial_t U_1^k + \partial_x F_{Euler}(U_1^k) + \partial_x F_{vis}(U_1^k) = 0 & \text{in } \Omega_1, \\ U_1^k = g & \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ U_1^k = U_2^{k-1} & \text{on } \Gamma_1, \end{cases}$$

$$\begin{cases} \partial_t U_2^k + \partial_x F_{Euler}(U_2^k) + \partial_x F_{vis}(U_2^k) = 0 & \text{in } \Omega_2, \\ U_2^k = g & \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ U_2^k = U_1^k & \text{on } \Gamma_2. \end{cases}$$

If the boundary conditions are Robin type, the second step of the algorithm changes:

1. Choose the initial solution for each sub-domain,
2. For $k = 1, 2, ...$ solve

$$\begin{cases} \partial_t U_1^k + \partial_x F_{Euler}(U_1^k) + \partial_x F_{vis}(U_1^k) = 0 & \text{in } \Omega_1, \\ \partial_{n_1} U_1^k + \lambda_1 U_1^k = g & \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ \partial_{n_1} U_1^k + \lambda_1 U_1^k = \partial_{n_1} U_2^{k-1} + \lambda_1 U_2^{k-1} & \text{on } \Gamma_1, \end{cases}$$

$$\begin{cases} \partial_t U_2^k + \partial_x F_{Euler}(U_2^k) + \partial_x F_{vis}(U_2^k) = 0 & \text{in } \Omega_2, \\ \partial_{n_2} U_2^k + \lambda_2 U_2^k = g & \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ \partial_{n_2} U_2^k + \lambda_2 U_2^k = \partial_{n_2} U_1^k + \lambda_2 U_1^k & \text{on } \Gamma_2. \end{cases}$$

The non-overlapping alternating Schwarz or, from a numerical point of view the non-overlapping multiplicative Schwarz method, is equivalent to a block Gauss-Seidel splitting method.

2.1.2 Parallel Schwarz Algorithm

Given Dirichlet type boundary conditions the Parallel Schwarz Algorithm is given by:

1. Choose the initial solution for each sub-domain,
2. For $k = 1, 2, ...$ solve

$$\begin{cases} \partial_t U_1^k + \partial_x F_{Euler}(U_1^k) + \partial_x F_{vis}(U_1^k) = 0 & \text{in } \Omega_1, \\ U_1^k = g & \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ U_1^k = U_2^{k-1} & \text{on } \Gamma_1, \end{cases}$$

$$\begin{cases} \partial_t U_2^k + \partial_x F_{Euler}(U_2^k) + \partial_x F_{vis}(U_2^k) = 0 & \text{in } \Omega_2, \\ U_2^k = g & \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ U_2^k = U_1^{k-1} & \text{on } \Gamma_2, \end{cases}$$

We remark that the difference between alternating Schwarz method and the parallel Schwarz method is the second transmission condition which uses an old iteration instead of the updated one. For a one direction partitioning, the algorithm can be generalized as follows:

1. Choose the initial solution for each sub-domain,
2. For each time instant $k = 1, 2, ...$
   - Solve for each cell $i = 1, ...$

$$\begin{cases} \partial_t U_i^k + \partial_x F_{Euler}(U_i^k) + \partial_x F_{vis}(U_i^k) = 0 & \text{in } \Omega_i, \\ U_i^k = g & \text{on } \partial\Omega_i \backslash (\Gamma_{left}^i \cup \Gamma_{right}^i), \\ U_i^k = U_{i-1}^{k-1} & \text{on } \Gamma_{left}^i, \\ U_i^k = U_{i+1}^{k-1} & \text{on } \Gamma_{right}^i, \end{cases}$$

where Ω_{i-1} and Ω_{i+1} are the neighbours sub-domains of Ω^i, $\Gamma^i_{left} = \partial\Omega_i \cap \Omega_{i-1}$ and $\Gamma^i_{right} = \partial\Omega_i \cap \Omega_{i+1}$.

The parallel Schwarz method for non-overlapping domains is equivalent to a block-Jacobi splitting method. The above algorithms have good convergence properties if the overlapping region is sufficiently large. The larger the overlap region, the faster the algorithm will converge. In this work we study domain decomposition methods for overlapping sub-domains with minimal overlapping region, which is the stencil size. Therefore, we also facilitate our tasks by choosing the same overlapping interval. One can increase the overlap region for one sub-domain, or only in one direction to obtain faster convergence. Because this technique is problem oriented, we decided to fix the overlap region to the size of the stencil for each numerical scheme. Otherwise, the overlap size equals the order of the numerical method in space. In this case, the first and last cells of one sub-domain will always have the necessary values to be computed.

We remark that both algorithms can be used for parallel computing. The parallel computation of the alternating algorithm must be done in two steps using alternatively sub-domains that do not touch each others and thus do not need to communicate. A technique to separate the sub-domains is the use of black-red colouring: all sub-domains that do not communicate have the same colour. This technique also avoids access and update conflicts when more than two regions overlap. Other improved algorithms that accelerate the convergence of the Schwarz methods have been proposed, such as the restrictive additive Schwarz algorithm and the two-level additive Schwarz. Another significant improvement can be achieved by modifying the transmission conditions. The resulting methods are called optimised Schwarz methods. The drawbacks of the overlapping techniques are: the sensitivity to strong discontinuities of coefficients, and the duplicate work on overlapping regions that can lead to significant computational cost for a large number of sub-domains. Moreover, it seems that they do not converge for all type of PDEs, for example the indefinite Helmholtz equation ([35]).

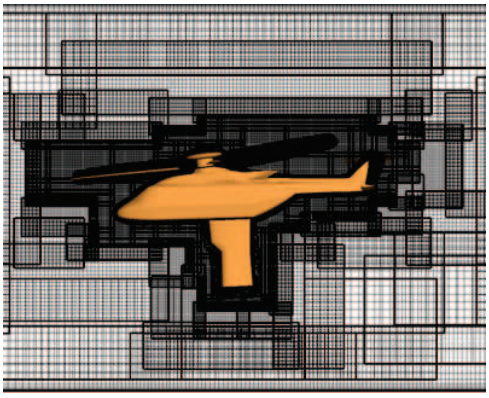


Figure 2.3: Helicopter virtual model generated with chimera method at Onera

One way to solve time depending systems consists in discretising the equations uniformly in time, then using decomposing techniques based on the presented Schwarz methods to solve the resulting semi-discretised in time system. In practice, we are often working with large computational domains, as on fig.2.3, where only a small part is highly interactive and a wide region of the domain is close to equilibrium state. Usually, the sub-domains are balanced in space so that each processor finishes the simulation at the same moment, and the computation is done, on each sub-domain with the same time step (global time step). This means that the part of the simulation domain which is not dominated by strong non-linearities is solved with a much higher precision than is needed. Some sub-domains are over-solved. Moreover, communication between sub-domains is synchronised at the end of each time step so a small amount of data (at least twice the size of the overlapping region) needs to be exchanged at every time step, which

can also be very costly. The time step is global, and computed sequentially, but the sub-domain close to equilibrium state converges in fewer iterations and it is less costly. Still, it has to wait for the high reactive sub-domain to end in order to continue the simulation. The iteration in time must be synchronised over all sub-domains, thus it proceeds only when each sub-domain has converged, which can be very penalizing. To avoid this loss of efficiency and optimize the computational cost, the time step should be computed locally and the distribution of flow in sub-domains should take into consideration several factors: closeness to equilibrium region, strong non-linearities region and time step influence. Another technical problem that can be avoided computing locally the time step is the asynchronous work of the parallel processors in use due to different latencies.

2.2 Schwarz Waveform Relaxation (SWR) Method

The Waveform Relaxation method or the Simple Dynamic Iteration scheme has first been proposed by an electrical engineering group at the University of California, Lelarasmee et al [62], in the context of integrated electrical simulations. The main idea is to use the relaxation method to exploit the latency of the large-scale circuits. The circuit is partitioned into sub-circuits, completed with artificial sources (as seen on fig.2.4) and independently compute each one of them with adaptable time steps for the entire time interval.

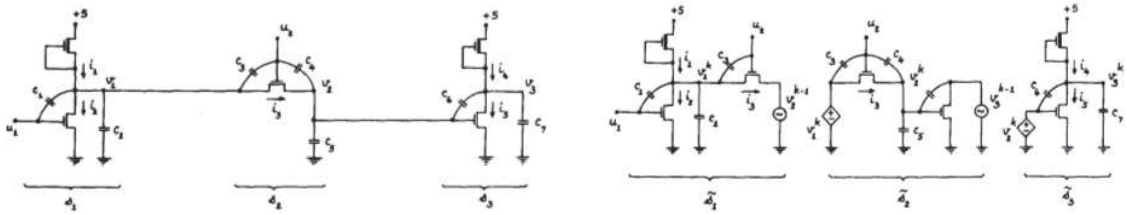


Figure 2.4: An MOS dynamic shift register (left) and its decomposition (right).

$$\begin{cases} \frac{dy^{n+1}}{dt} = f(t, y^n) \\ y^{n+1}(0) = y_0 \end{cases}, \quad y : [0,T] \to \mathbb{R}^d \text{ and } y^n \text{ is a vectorial sequence of } y. \tag{2.2}$$

$$\begin{cases} \frac{dy_1^{n+1}}{dt} = f(y_1^{n+1}, y_2^n, y_3^n) \\ y_1^{n+1}(0) = y_{01} \end{cases}, \begin{cases} \frac{dy_2^{n+1}}{dt} = f(y_2^n, y_2^{n+1}, y_3^n) \\ y_2^{n+1}(0) = y_{02} \end{cases}, \begin{cases} \frac{dy_3^{n+1}}{dt} = f(y_1^n, y_2^n, y_3^{n+1}) \\ y_3^{n+1}(0) = y_{03} \end{cases}. \tag{2.3}$$

The discretised system to solve has the form of (2.2). When the circuit is partitioned (fig.2.4, right), each unknown variable is assigned to an equation (represented on (2.3) for three partitions) and each equation is solved as an independent system using either the Gauss-Seidel or the Gauss-Jacobi relaxation. Inside one iteration of Gauss-Seidel, the waveform solution at intermediate time steps is immediately used to update the neighbour solution. For Gauss-Jacobi, the solution is updated at the end of each iteration. Based on simulations and experiments, Lerasmee et al [62] noticed that, for a large circuit or for large time intervals the data storage can become very large as well, and that a good initial guess can, considerably, accelerate the convergence.

The obvious advantage of this method is not only that it allows the parallel computation of smaller sub-systems, but also the use of different time steps for different sub-systems. The method works very well for electrical circuits, where the system coupling systems is done naturally and the coupling occurs only for short time intervals. It can be very slow in the case of strong coupling between two systems. Several acceleration techniques exist, as for example the multigrid or preconditioning method (see [11] for details or more references). Another important observation is that the convergence of the waveform relaxation approach slows down

when t increases toward the time interval limit. Thus, it is necessary to control the time window length. An idea is to impose the number and the length of time windows inside the time interval of the simulation. But, this technique does not take into account the evolution of the waveforms and a non-uniform repartition seems more adapted. Up to our knowledge, there is no efficient algorithm able of automating the choice of the number of time windows and their length, especially due to the large variety of the applications. Although, we can recall the effort of Burrage and Dyke [12] where they concluded that the results using the proposed techniques of adapting the time window are not conclusive, since it can result in better efficiency for several particular cases, and less efficiency for other particular cases.

In 1994, in his technical report [5], M. Bjorhus analyses the dynamic iteration method and gives a proof of its super-linear convergence. One year later he writes a second technical report [6], on the Semi-discrete sub-domain iteration for hyperbolic systems, another similar technique to decouple systems of equations. He suggests, after decomposing the space domain, to decompose the time into small intervals or windows and to solve each sub-problem over one window, in parallel using an iterative method. He mentioned that the size of the window can be variable, that a larger window can save computational cost, but also that the window length can influence the convergence rate. In parallel, in 1995, Jeltsch and Pohl [58] extend a matrix multi-splitting method (see [81]) with overlap for solving large systems of ordinary differential equations which is equivalent to a waveform relaxation method for special set of parameters. Motivated by the work of Bjorhus and the interesting algorithm on multi-splitting with overlap proposed by Jeltsch and Pohl, M. Gander and A. Stuart [46] propose a multi-splitting formulation on overlapping sub-domains combined with a waveform relaxation algorithm in space-time for the heat equation. They prove linear convergence of the algorithm on an infinite time interval and that the convergence rates remains robust for refined meshes. In the Proceedings of the 10th International Conference on Domain Decomposition, Gander [38] introduces the method as the Overlapping Schwarz Waveform Relaxation method (SWR) and shows that the method holds also for parabolic equations. He concluded with a very essential property of the SWR algorithm: *the super-linear convergence rates do not depend on the number of the sub-domains and thus there is no coarse mesh needed to avoid deterioration of the algorithm.* Since then, the SWR method has been intensively studied and developed, mostly in the linear case by M. Gander, L. Halpern and others [43, 40, 73, 49].

The main advantage of the SWR is that there is no need to exchange information at every time step (as for the classical Schwarz decomposition), nor only at the end of the time computation (as in the case of the waveform relaxation methods, where we can loose the convergence for large time intervals), but to exploit both methods using time windowing techniques that do not degrade the solution and exchange less information between processors. The type of the interface condition between two sub-domains influences the efficiency of the SWR method. We can find many studies using classical interface conditions of Dirichlet type, but also optimised SWR algorithms where the classical interface conditions are replaced by transparent boundary conditions, approximations of the transparent boundary conditions, or Robin type boundary conditions, see [43, 40, 73]. In the paper of V. Martin, [73], the influence of the the time window lengths over linear systems is also discussed. SWR has also been extended to non-linear cases thanks to the Newton scheme and successfully applied to the reactive transport equations in the PhD. thesis of F. Haeberlein [49] and to the incompressible Navier–Stokes equations by Audusse and al [2].

We can resume the SWR method in two main steps:

1. space discretisation;
2. waveform relaxation of the system (compute the system in parallel using different time steps).

SWR algorithm converges linearly over long time intervals for diffusive problems and at least linearly over short time intervals (see [53] for results of the convergence of SWR for parabolic

equations (super-linear convergence for finite time), of the optimized SWR for parabolic equations of for other types of PDEs).

Let us reconsider the same overlapping decomposition in two sub-domains as for the Schwarz methods (fig.2.1, left). For the 1D Navier–Stokes time dependent problem and a decomposition on two sub-domains, the SWR algorithm can be represented as follows:

1. Choose the initial solution for each sub-domain;
2. For $k = 1, 2, ...$ solve

$$\begin{cases} \partial_t U_1^k + \partial_x F_{Euler}(U_1^k) + \partial_x F_{vis}(U_1^k) = 0 & \text{in } \Omega_1 \times]0,T], \\ U_1^k = U_0 & \text{on } \Omega_1 \times \{0\}, \\ U_1^k = g & \text{on } (\partial\Omega_1 \backslash \Gamma_1) \times [0,T], \\ U_1^k = U_2^{k-1} & \text{on } \Gamma_1 \times [0,T], \end{cases}$$

$$\begin{cases} \partial_t U_2^k + \partial_x F_{Euler}(U_2^k) + \partial_x F_{vis}(U_2^k) = 0 & \text{in } \Omega_2 \times]0,T], \\ U_2^k = U_0 & \text{on } \Omega_2 \times \{0\}, \\ U_2^k = g & \text{on } (\partial\Omega_2 \backslash \Gamma_2) \times [0,T], \\ U_2^k = U_1^{k-1} & \text{on } \Gamma_2 \times [0,T], \end{cases}$$

where u_0 is the initial guess and g the boundary condition over the time interval.

In the following work, the windowing technique has been adopted: the entire time interval $[0,T]$ is divided in windows $[0,T^1], [T^1,T^2], ..., [T^{l-1},T]$ that can have different lengths. The length of one window is a problem oriented choice in such a way that the convergence rate is not diminished. A good criterion for choosing the length of one window should depend on the largest local time step. The algorithm must converge inside each window before passing to the next one, involving iterations over the entire window, and not only over one time step (as for example for the parallel Schwarz scheme). For a chosen time window, $\Delta T = \alpha \max(\Delta t_1^0, \Delta t_2^0)$ where Δt_1^0 and Δt_2^0 are the time steps computed locally in Ω_1, respectively Ω_2, and α a chosen parameter, one can illustrate the SWR iterations over one window, as on fig.2.5.

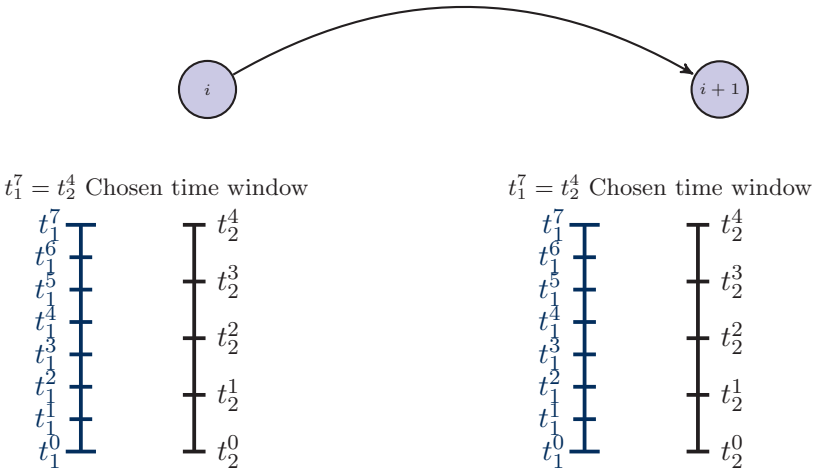


Figure 2.5: Possible configuration of the SWR classical iterations i and $i+1$ for a chosen time window. Once the length of a time window is chosen, each sub-domain evolves with equal size time steps.

2.3 Adaptive Schwarz Waveform Relaxation (ASWR) Method

Within the SWR iterative process, we propose two adaptive time stepping techniques to improve the scheme consistency. There is no existing algorithm capable to predict the optimal

time window length. This one seems to differ from one particular case to another. We have only few clues on how a time window should be chosen.

- Inside a time window, the errors are allowed to propagate in time and they are corrected only after the end of one iteration over the entire time window. If the error is too large, the correction will require many iterations and the algorithm will become inefficient.
- The time step should consider the length of the initial time step. About the initial time steps we know that they are computed in such a way to satisfy a CFL type condition of mathematical stability (for explicit schemes) or numerical stability (implicit schemes).

The whole idea of SWR is to take different time steps inside each sub-domain. This leads to a non-coincident time discretisation and the need of an interpolation on the artificial boundary. In order to be consistent with a second order scheme in time and in space we choose a quadratic interpolation. Thus, we get another clue over a time window length: it should contain at least two time steps to enable the quadratic interpolation. Based on the previous remarks we propose two constraints over the window length. For two sub-domains a)$\Delta T = \alpha \min(\Delta t_0^1, \Delta t_0^2)$ and b)$\Delta T \geq 2 \max(\Delta t_0^1, \Delta t_0^2)$, thus $\alpha \geq 2$. Yet, we do not have any information over the upper bound of α.

One level local adaptivity

Inside one time window, the entire Navier–Stokes system of equations, is to be solved completely locally. Then the stability condition can also be locally considered. Once the window size is chosen, we propose to locally recompute a new time step and renew its computation after each progression in time. The time-stepping technique depends then on the stability (mathematical and numerical) and on the constraints $\alpha \geq 2$. This new flexibility (on fig.2.6) improves the consistency of the scheme and renders automatic the discretisation of one time window.

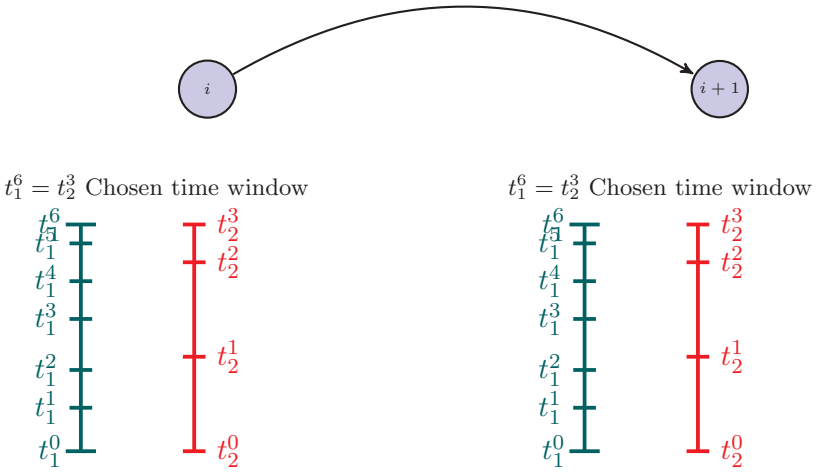


Figure 2.6: Example of adaptive steps inside a time window for two iterations i and $i+1$. Once the length of a time window is chosen, each sub-domain evolves with adaptive time steps. Each time step is recomputed to satisfy a local stability criteria. Yet, the last time step can be smaller to not exceed the window length. The time discretisation is then fixed for all window iterations.

Two levels local adaptivity

Based on the same ideas of preserving local stability and consistence, we propose to automatically adapt a second level inside the SWR algorithm. This time, by adaptivity we mean

that, for each SWR iteration, as we improve the coupling conditions we adapt the time step to satisfy the CFL condition (for explicit schemes) and ensure stability of our method, thus different time steps in each sub-domain, inside each time window and at each iteration. This extension of the SWR flexibility can be represented as on fig.2.7.

Techniques of adapting space steps or local refinement are often used in order to correctly discretise the geometry of the problem to solve. In what concerns the time dimension, local time stepping is less used even though it results in efficiently allocating computational resources and gives solutions more reliable. We refer to the recent article of M. Gander and L. Halpern [41] for a survey over the research on time-stepping over the last two decades.

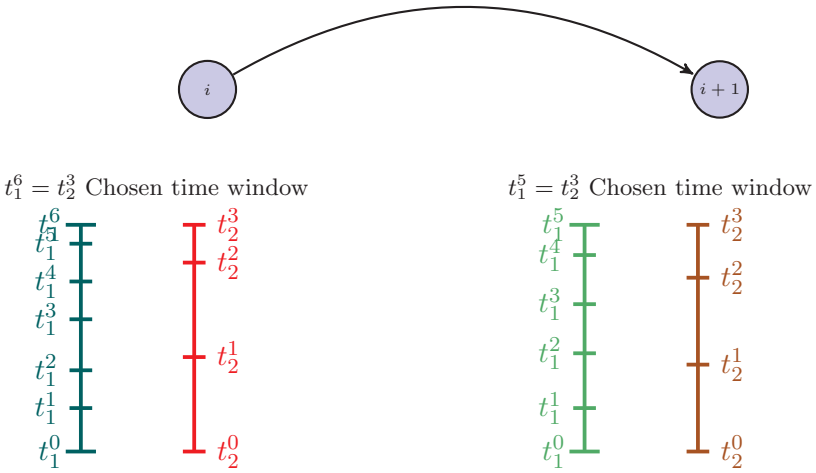


Figure 2.7: Example of adaptive steps inside a time window and at each iteration, i and $i+1$. Once the length of a time window is chosen, each sub-domain evolves with adaptive time steps. Each time step is recomputed to satisfy a local stability criteria not only at the first iteration, but at each Schwarz iteration. Thus, different time window discretisations at each iterations.

We investigate the effects of this improvement on the numerical (computational) stability. Independently of the chosen numerical scheme, the time step needs a controlling criteria, a criteria of acceptance in order to become efficient. For explicit schemes, the time step is chosen in such a way to respect the continuous stability criteria depending on the CFL number. One would want to impose a minimal length of the time step to reduce the numerical errors. It means that the time step needs, not only to respect the physical modelling properties, but also to minimise the computational errors. If the time step is chosen too small, a sub-domain may be over-solved, numerical errors can appear due to round-off, unnecessary computational work can be done. Otherwise, a too large time step can lead to inaccurate results. Adapting the time step allows controlling the accuracy of the simulation and improves its efficiency.

We motivate our choice to locally recompute the time steps inside each window, but also at each iteration by the improvement added to the solution from one iteration to another. We iterate over one time window until the numerical convergence is achieved, by numerical convergence we mean that the error between the solutions found at two consecutive iterations in the end of one time window is smaller than a chosen tolerance. After each iteration we proceed to the improvement of the interface condition in each sub-domain. An improved interface condition could lead to a completely different necessary time step to accomplish the continuous stability criteria (for explicit schemes) thus the necessity to locally recompute the time step.

Discussions

The explicit schemes are easy to implement and to parallelise, they give a solution low

cost consuming per time step and constant per iteration. But, the computation is limited in time due to the extremely small time step required for stability reasons. It is the reason for which they are inefficient for computations of steady problems. The local computation of the time steps introduced by SWR schemes allows a time prolongation for more stiff cases. Yet, the nature of the simulations requires the introduction of more complex approaches, such as implicit schemes. The implicit schemes are very adaptable to the steady-state cases and stable over a wide range of time steps, sometimes even unconditionally. In their original state they are difficult to implement and parallelise, being very costly per time step. For unsteady cases they can deteriorate the solution for large time steps. In the next section we shall apply the SWR methods to the implicit classical schemes and discuss their efficiency.

Inside one sub-domain the time step is always computed locally.

2.4 Schwarz Methods applied to implicit solvers

Implicit schemes have been intensively used and studied especially for steady cases, they are free of any stability bound, but often condemned for their large arithmetic operation counts and thus for their high computational cost. Different from the classical Alternating Schwarz method, the idea of preconditioning a non-linear system of equations with a Schwarz based domain decomposition method was strictly introduced for parallel processing purpose [14]. The implicit class of methods for solving non-linear systems are characterised by multi-level algorithms. The domain decomposition techniques for the implicit solvers are defined by the level of their application. We emphasize three main classes of decomposition methods, Newton-Schwarz, Schwarz-Newton and SWR, and discuss about ways to improve their efficiency. In this section, all algorithms are based on the same time discretisation (second order implicit Backward Differentiation Formula), the non-linear problem is solved with the Newton method and linear problems are solved directly ($(\mathcal{L}+\mathcal{D})\mathcal{D}^{-1}(\mathcal{D}+\mathcal{U})$ factorisation).

The most basic way to solve a problem in parallel is to only use a partitioning technique. The classical partitioning method for implicit solvers consists in applying a global Newton linearisation, then dividing the linear system in several local overlapping subsystems that we can solve in parallel. After one iteration we rebuild the global problem through Dirichlet coupling conditions and we continue the Newton iterations. We call this kind of coupling an explicit one, since no supposition on the coupling is made. This technique is equivalent to a domain decomposition technique with only one iteration over each sub-domain.

Newton-Partitioning Algorithm:

- Semi-discretisation in time,
- Linearisation (Newton),
- Space partitioning,
    - ⋆ Solve the local linear system.

The classical non-linear domain decomposition method consists in semi-discretising uniformly in time the system, in applying a global Newton linearisation, then dividing the linear system in several local overlapping subsystems that we can solve in parallel. This algorithm is referred to as the Newton-Schwarz algorithm and it was first introduced by Cai and co [14, 18, 15, 59, 60] as a joining technique between the Newton-Krylov method and the Krylov-Schwarz method. Their motivation was that in a Newton-Krylov method an ill-conditioned Jacobian will require an unacceptable number of iterations and that the Krylov-Schwarz preconditioning is locally introduced on each sub-domain and can adapt to the time-evolving ill-conditioning of the linear system.

Newton-Schwarz Algorithm:

- Semi-discretisation in time,
- Linearisation (Newton),
- Space Schwarz DDM,
  - ⋆ Solve the local linear system.

In some cases, one Schwarz iteration is sufficient to achieve convergence of Newton to the solution of the problem and the algorithm is equivalent to the above Newton-Partitioning algorithm presented in section 2.4. Space decomposition and linearisation are independent. The next idea is to first do the decomposition and then solve in each sub-domain the non-linear system. This algorithm was also introduced and intensively studied by Cai and co. [14, 16, 17, 72], but using a different linear solver.

Schwarz-Newton Algorithm:

- Semi-discretisation in time,
- Space Schwarz DDM,
  - ⋆ Solve the local non-linear system.

To achieve full speed-up performance, a SWR method is used, as it allows local space and time stepping. The whole time interval of study is split into sub-intervals or time windows, then the space is decomposed into sub-domains. For each time window the space-time Navier-Stokes equations are solved in each sub-domain in parallel. Boundary conditions are transmitted at the end of the time window.

SWR Algorithm:

- Schwarz DDM over time windows,
- For each sub-domain:
  - ⋆ Semi-discretisation in time,
  - ⋆ Solve the local linear system.

SWR uses time windowing techniques that do not degrade the solution and exchanges less information between processors. After each iteration we proceed to the improvement of the interface condition in each sub-domain. This can lead to a completely different time step to satisfy either a stability criteria (for explicit schemes) or an accuracy bound, both based on the CFL number, thus the necessity to locally recompute the time step which is an improvement of the classical SWR algorithm. In this section we propose, within the SWR iterative process, an adaptive time stepping technique to improve the scheme consistency, thus different time steps in each sub-domain and inside each time window.

These methods are supposed to be independent of the type of equation that are applied to, yet they do not necessary lead to good performance. Poor quality transmission conditions between artificial boundaries can degrade the convergence. In the next section we discuss the influence of the interface conditions on the convergence. We propose different ways of implementing first order accuracy interface conditions (Robin type) and discuss over their efficiency through different choices of parameters.

2.5 Transmission conditions on artificial boundaries

The initial and boundary conditions are given such that the overall accuracy of the solution is not diminished. Usually, they are at least one-order lower accuracy than at inner points (see [48] for mixed initial boundary value problems), but directly act on a very small amount of points. A domain decomposition method introduces artificial boundaries and it is up to the user

to decide their type and order in such a way that the overall accuracy and the convergence are not degraded. It is natural to think that more sub-domains involve more artificial boundaries and lead to an increased number of points computed with lower accuracy and thus to low overall accuracy and poor convergence.

The Dirichlet interface condition, meaning of zero order is quite easy to compute and preferred in most industrial codes. But, higher order boundary conditions can improve the convergence for the same overlap or reach the same convergence for smaller overlap. In his third paper on the Schwarz alternating method [66], P.L. Lions proposes a non-overlapping algorithm by introducing the Robin type interface conditions instead of the Dirichlet type which he considers restrictive as it needs some kind of overlap to work. It was the beginning of a wide research attempt to improve convergences by improving the transmission conditions between artificial boundaries in the domain decomposition field. The Robin condition, also known as the Fourier condition or third type boundary condition (or impedance boundary conditions in the electromagnetic field) is a weighted linear combination of a Dirichlet and a Neumann condition. The Dirichlet and Neumann conditions are considered as the first and the second type boundary conditions. Using Robin type boundary condition improves convergence on many kinds of applications, but the great challenge is imposed by the difficulties to find the best Robin parameter and generalise its expression. We refer to the literature for applications which uses Robin transmission conditions, such as the advection-diffusion-reaction with non-overlapping sub-domains [42], the Poisson equation for non-overlapping sub-domains with cross points [45], the Laplace equation for overlapping sub-domains and coarse grid [29, 30]. Others have successfully tried other techniques of high order transmission conditions like the absorbing boundary conditions [73, 44, 52] to improve the convergence of linear shallow water equations, Laplace and Helmholtz equations or advection-diffusion equation.

In our case the whole domain decomposition strategy is based on the use of overlapping ghost cells (fictitious overlap) to conserve the physical boundary and the interface conditions so that there would be no change inside the numerical algorithm. The boundary conditions need to be handled carefully since they ensure the well posedness of a system of equations. If the Dirichlet boundary condition fits very well our strategy, we wonder if Robin type interface conditions can also be adapted to the use of ghost cells and how should we proceed to improve convergence and reduce computational costs.

In this section we study the 1D Cauchy problem issued from the one dimensional Navier–Stokes system of equations with Dirichlet and/or Robin boundary condition. We keep the same notations as in the previous sections and we recall the 1D Navier–Stokes system of equations defined on the domain Ω:

$$\left\{\begin{array}{ll} \frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} & = 0 \ \text{ in } \ \Omega \times [0,T], \\ \frac{\partial \rho u}{\partial t} + \frac{\partial (\rho u^2 + p)}{\partial x} - \frac{\partial \tau}{\partial x} & = 0 \ \text{ in } \ \Omega \times [0,T], \\ \frac{\partial \rho E}{\partial t} + \frac{\partial (u(\rho E + p))}{\partial x} - \frac{\partial (\tau u - q)}{\partial x} & = 0 \ \text{ in } \ \Omega \times [0,T], \end{array}\right.$$

where the deformation tensor τ becomes $\tau = (-2\lambda)\frac{\partial u}{\partial x}$. The conservative vector U is restricted to three components $(\rho, \rho u, \rho E)$. We are mostly interested on the interface conditions and data transfer between sub-domains. Therefore, to ensure the well posedness of the global problem we impose first type or Dirichlet boundary condition on its physical boundaries and we will change the artificial boundary conditions from first type to a mix between the first and third (Robin) type conditions and to third type conditions.

Let us fix the space domain Ω to the interval $[a, b]$ and $[0, T]$ the time domain. The Cauchy problem issued from the condensed form of the 1D Navier–Stokes equations (2.5) and Dirichlet

boundary condition has the following form:

$$\text{Find U that satisfies} \begin{cases} \partial_t U + \partial_x F_{Euler}(U) + \partial_x F_{vis}(U) = 0 & \text{in } \Omega \times [0,T], \\ U = U_0 & \text{on } \Omega \times \{0\}, \\ U = U_D & \text{on } \partial\Omega \times [0,T], \end{cases} \tag{2.4}$$

where F_{Euler} is the convective flux and F_{vis} if the diffusive flux, defined in section 1.1.1. Let us suppose that meshes are coincident. Given Dirichlet and Robin type boundary conditions a generalised parallel Schwarz algorithm for two sub-domains can be structured as follow:

- split the original domain Ω into two overlapping sub-domains, $\Omega = \Omega_1 \cup \Omega_2$;
- denote Γ_1 and Γ_2 the fictitious layers which contain the interface boundaries;
- separate the Dirichlet and/or the Robin boundaries: $\Gamma_1 = \{\Gamma_1^D, \Gamma_1^R\}$ and $\Gamma_2 = \{\Gamma_2^D, \Gamma_2^R\}$. This separation is possible because of the chosen numerical scheme where the stencils have different size for the computation of the convective (three points stencil) and diffusive fluxes (five points stencils);
- Choose the initial solution;
- For $k = 1, 2, ...$ solve

$$\begin{cases} \partial_t U_1 + \partial_x F_{Euler}(U_1) + \partial_x F_{vis}(U_1) = 0 & \text{in } \Omega_1, \\ U_1 = U_D & \text{on } \partial\Omega_1 \setminus \Gamma_1, \\ U_1^k = U_2^{k-1} & \text{on } \Gamma_1^D, \\ \dfrac{\partial U_1^k}{\partial n_1} + \lambda_1 U_1^k = \dfrac{\partial U_2^{k-1}}{\partial n_1} + \lambda_2 U_2^{k-1} & \text{on } \Gamma_1^R, \end{cases}$$

$$\begin{cases} \partial_t U_2 + \partial_x F_{Euler}(U_2) + \partial_x F_{vis}(U_2) = 0 & \text{in } \Omega_2, \\ U_2 = U_0 & \text{on } \partial\Omega_2 \setminus \Gamma_2, \\ U_2^k = U_1^{k-1} & \text{on } \Gamma_2^D, \\ \dfrac{\partial U_2^k}{\partial n_2} + \lambda_2 U_2^k = \dfrac{\partial U_1^{k-1}}{\partial n_2} + \lambda_1 U_1^{k-1} & \text{on } \Gamma_2^R, \end{cases} \tag{2.5}$$

where n_1 (respectively n_2) is the unit external normal to Ω_1 (respectively Ω_2), λ_1, λ_2 are the Robin parameters left to define.

In particular, for $\Omega = [a, b]$ the main lines of the algorithm are the following:

- $\Omega = \Omega_1 \cup \Omega_2$ becomes $[a, b] = [a, c + \theta\Delta x] \cup [c - \theta\Delta x, b]$. For a viscous flow we always suppose an overlap θ of larger size than the stencil size. For a second order scheme 2 cells play the role of ghost cells. If the overlap equals the stencil size we call it fictitious overlap since its role is reduced to save values of local boundary conditions.
- denote $\Gamma_1 = \{c + (\theta - 1)\Delta x, c + \theta\Delta x\}$ and $\Gamma_2 = \{c - \theta\Delta x, c - (\theta - 1)\Delta x\}$.
- separate the Dirichlet and/or the Robin boundaries For mixed boundary conditions, the most exterior cells contains the Robin boundaries and the most interior ones the Dirichlet condition.
- For $k = 1, 2, ...$ solve (2.5).

The choice of this decomposition is motivated by our Finite Volume discretisation to solve the Navier–Stokes system of equations. In section 1.2.3 we studied the Euler fluxes and gave different numerical approach to find their value. We noticed that, independently of the chosen scheme, the computation of the divergence of the convective flux $F_{Euler}(U_i)$ depends on the known values in the previous cell. Actually, we must compute only first order derivatives

$$(F_{Euler})_x = \left(\frac{\partial \rho u}{\partial x}, \frac{\partial(\rho u^2 + p)}{\partial x}, \frac{\partial u(\rho E + p)}{\partial x} \right)^t,$$

for schemes that are at most second order accuracy. In order to compute the 1D diffusive flux

$$(F_{vis})_x = \left(0, \frac{\partial}{\partial x}(\frac{2}{3}\frac{\mu}{Re}\frac{\partial u}{\partial x}), \frac{\partial}{\partial x}(\frac{2}{3}\frac{\mu}{Re}u\frac{\partial u}{\partial x} + q)\right)^t,$$

second order derivatives are defined as in 1.2.4.

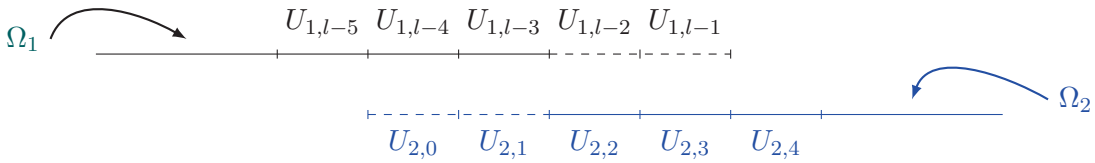


Figure 2.8: Two overlapping sub-domains with overlap of size $\theta = 2$, the stencil size. The ghost cells are represented by dashed lines.

We consider $a = x_{1,-\frac{1}{2}} < x_{1,\frac{1}{2}} < x_{1,\frac{3}{2}} < ... < x_{1,l-\frac{1}{2}} = c + \theta\Delta x$ the space discretisation of $\Omega_1 = [a, c+\theta\Delta x]$ and $c-\theta\Delta x = x_{2,-\frac{1}{2}} < x_{2,\frac{1}{2}} < x_{2,\frac{3}{2}} < ... < x_{2,m-\frac{1}{2}} = b$ the space discretisation of $\Omega_2 = [c-\theta\Delta x, b]$. On fig.2.8 we have highlighted the interface between the two sub-domains. When computing the first cell proper to the domain Ω_2, $U_{2,2}$ (we exclude the ghost cells), the convective flux needs the value of $U_{2,1}$, and the diffusive flux needs the value of $U_{2,0}$ and $U_{2,0}$. When computing the last cell proper to the domain (not a ghost cell) Ω_1, $U_{1,l-3}$, the convective flux needs the values of $U_{1,l-1}$ and the diffusive flux the value of $U_{1,l-2}$ and $U_{1,l-1}$. In our algorithm convective fluxes and diffusive fluxes are treated separately and independently.

2.5.1 Dirichlet type boundary conditions

The concept of Dirichlet boundary condition is identical to a fixed boundary condition. In our case the concept is not exactly adopted, as values on the ghost cells are renewed after each iteration, see for example the 1D case presented on fig.2.9. We suppose that all values proper to one sub-domain are known (for $t = 0$ we give an initial solution) at time $t = k$. For minimal overlap, in order to compute both the convective and the diffusive fluxes at time $t = k + 1$, the following exchange of data between the two sub-domains is sufficient:

$$\Omega_1: \begin{cases} U^{k+1}_{1,l-2} &= U^k_{2,2}, \\ U^{k+1}_{1,l-1} &= U^k_{2,3}, \end{cases}$$

$$\Omega_2: \begin{cases} U^{k+1}_{2,0} &= U^k_{1,l-4}, \\ U^{k+1}_{2,1} &= U^k_{1,l-3}. \end{cases}$$

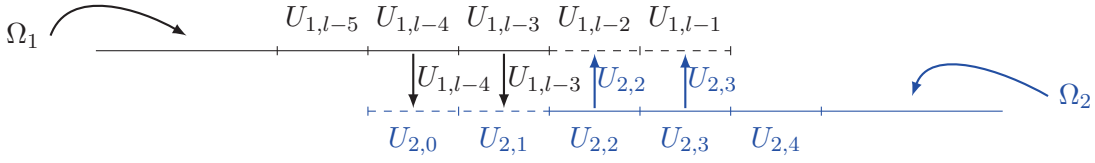


Figure 2.9: Fictitious overlapping sub-domains. Fill out of Dirichlet conditions

Identical treatment is done for both convective and viscous fluxes. Moreover, to solve the Euler equations one only needs one boundary condition, one ghost cell is sufficient to the

computation. The Dirichlet boundary type is the preferred one in the industrial codes due to its simplicity and to its direct application to the Parallel Schwarz method. If the overlap is larger than half the stencil size the Dirichlet boundary condition is similarly imposed. In the figures 2.15, 2.16, 2.18, 2.20, this condition (b)) is denoted as Dirichlet-Dirichlet.

2.5.2 Mixed Dirichlet/Robin boundary conditions

The first idea of transmission boundary condition comes from the observation that only the computation of the diffusive fluxes uses the most exterior ghost cells values. Thus, Robin type transmission conditions can be used to ensure the C^1 continuity of the diffusive solution. For minimal overlap, we begin by filling out the closest ghost cell to the domain with Dirichlet conditions (represented on fig.2.10).

$$\begin{aligned}\Omega_1: \quad & U^{k+1}_{1,l-2} = U^k_{2,2},\\ \Omega_2: \quad & U^{k+1}_{2,1} = U^k_{1,l-3}.\end{aligned}$$

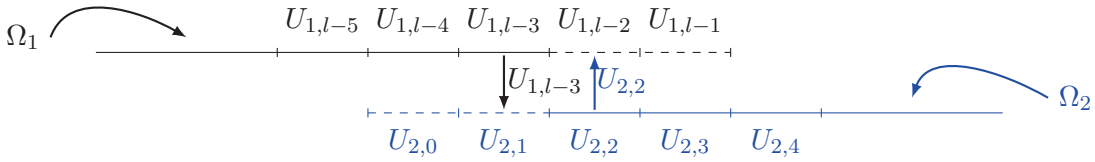


Figure 2.10: Fictitious overlapping sub-domains. Fill out of Dirichlet conditions

The next question is where should the Robin condition be imposed. We propose two different ways.

a) A first idea is to suppose that the Robin boundary condition is fixed on the exact interface between the two sub-domains. On fig.2.11 this is at the position $x_{1,l-2-\frac{1}{2}} = x_{1,l-\frac{5}{2}}$ on Ω_1 that is coincident with the position $x_{2,1+\frac{1}{2}} = x_{2,\frac{3}{2}}$ on Ω_2.

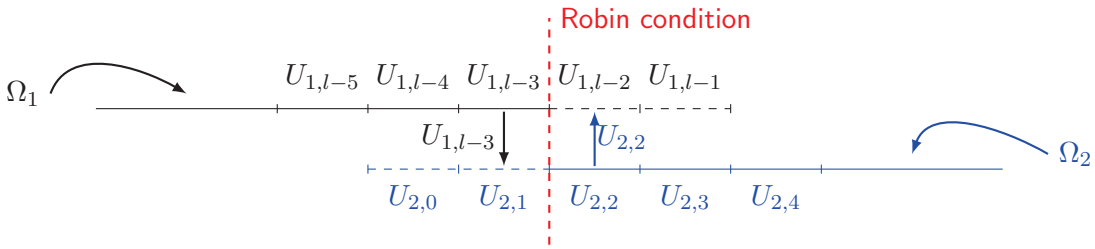


Figure 2.11: Case 1: Position of Robin boundary between two fictitious overlapping sub-domains used to fill out $U_{2,0}$ and $U_{1,l-1}$. The first ghost cells are filled with a value arising from Dirichlet type boundary condition.

$$\begin{aligned}\Omega_1: \quad & \frac{\partial U^k_{1,l-\frac{5}{2}}}{\partial n_1} + \lambda_1 U^k_{1,l-\frac{5}{2}} = \frac{\partial U^{k-1}_{2,\frac{3}{2}}}{\partial n_1} + \lambda_2 U^{k-1}_{2,\frac{3}{2}} \overset{noted}{=} g_2^{k-1},\\ \Omega_2: \quad & \frac{\partial U^k_{2,\frac{3}{2}}}{\partial n_2} + \lambda_2 U^k_{2,\frac{3}{2}} = \frac{\partial U^{k-1}_{1,l-\frac{5}{2}}}{\partial n_2} + \lambda_1 U^{k-1}_{1,l-\frac{5}{2}} \overset{noted}{=} g_1^{k-1}.\end{aligned}$$

At convergence $U^k = U^{k-1}$ and $\frac{\partial U^k}{\partial n} = \frac{\partial U^{k-1}}{\partial n}$. Even though we have centred the Robin interface on the common physical interface, the values to fill out are those of the most exterior ghost cells. We propose then to approximate the value of the conservative vector with a polynomial that crosses through three points on each sub-domain,

$$P_\Omega(x) = \alpha_0 x^2 + \alpha_1 x + \alpha_2. \tag{2.6}$$

Let us start with Ω_1, the target domain, and Ω_2 the source domain. We recall that, in this case, the value to fill out is $U_{1,l-1}$ (for simplicity we omit in the following the index k). When we refer to this situation we call the sub-domain Ω_1 the target and the sub-domain Ω_2 the source of the computation. In order to find the coefficients we impose that:

$$\begin{cases} P_{\Omega_1}(x_{1,l-3}) = U_{1,l-3}, \\ P_{\Omega_1}(x_{1,l-2}) = U_{1,l-2}, \\ P_{\Omega_1}(x_{1,l-1}) = U_{1,l-1}, \end{cases}$$

which is equivalent to a system of three equations with three unknowns $\alpha_{1,0}, \alpha_{1,1}$ and $\alpha_{1,2}$:

$$\begin{cases} \alpha_{1,0} x_{1,l-3}^2 + \alpha_{1,1} x_{1,l-3} + \alpha_{1,2} = U_{1,l-3}, \\ \alpha_{1,0} x_{1,l-2}^2 + \alpha_{1,1} x_{1,l-2} + \alpha_{1,2} = U_{1,l-2}, \\ \alpha_{1,0} x_{1,l-1}^2 + \alpha_{1,1} x_{1,l-1} + \alpha_{1,2} = U_{1,l-1}. \end{cases}$$

We note

$$\det_1 = (x_{1,l-3} - x_{1,l-2})(x_{1,l-2} - x_{1,l-1})(x_{1,l-3} - x_{1,l-1}),$$

and, by solving the system of equations we obtain:

$$\begin{cases} \alpha_{1,0} = \frac{1}{\det_1}\big[U_{1,l-3}(x_{1,l-2} - x_{1,l-1}) + U_{1,l-2}(x_{1,l-1} - x_{1,l-3}) + U_{1,l-1}(x_{1,l-3} - x_{1,l-2})\big], \\ \alpha_{1,1} = \frac{1}{\det_1}\big[U_{1,l-3}(x_{1,l-1}^2 - x_{1,l-2}^2) + U_{1,l-2}(x_{1,l-3}^2 - x_{1,l-1}^2) + U_{1,l-1}(x_{1,l-2}^2 - x_{1,l-3}^2)\big], \\ \alpha_{1,2} = \frac{1}{\det_1}\big[U_{1,l-3} x_{1,l-2} x_{1,l-1}(x_{1,l-2} - x_{1,l-1}) + U_{1,l-2} x_{1,l-3} x_{1,l-1}(x_{1,l-1} - x_{1,l-3}) \\ \qquad + U_{1,l-1} x_{1,l-3} x_{1,l-2}(x_{1,l-3} - x_{1,l-2})\big]. \end{cases}$$

The objective is to compute the value of $U_{1,l-1}$. On the source, to compute g_2, we first compute the polynomial P_{Ω_2} with the following constraints:

$$\begin{cases} P_{\Omega_2}(x_{2,1}) = U_{2,1}, \\ P_{\Omega_2}(x_{2,2}) = U_{2,2}, \\ P_{\Omega_2}(x_{2,3}) = U_{2,3}, \end{cases}$$

which is equivalent to:

$$\begin{cases} \alpha_{2,0} x_{2,1}^2 + \alpha_{2,1} x_{2,1} + \alpha_{2,2} = U_{2,1}, \\ \alpha_{2,0} x_{2,2}^2 + \alpha_{2,1} x_{2,2} + \alpha_{2,2} = U_{2,2}, \\ \alpha_{2,0} x_{2,3}^2 + \alpha_{2,1} x_{2,3} + \alpha_{2,2} = U_{2,3}. \end{cases}$$

We note $\det_2 = (x_{2,1} - x_{2,2})(x_{2,2} - x_{2,3})(x_{2,1} - x_{2,3})$ and, by solving in the same manner the system of equations we obtain:

$$\begin{cases} \alpha_{2,0} = \frac{1}{\det_2}\big[U_{2,1}(x_{2,2} - x_{2,3}) + U_{2,2}(x_{2,3} - x_{2,1}) + U_{1,3}(x_{2,1} - x_{2,2})\big], \\ \alpha_{2,1} = \frac{1}{\det_2}\big[U_{2,1}(x_{2,3}^2 - x_{2,2}^2) + U_{2,2}(x_{2,1}^2 - x_{2,3}^2) + U_{2,3}(x_{2,2}^2 - x_{2,1}^2)\big], \\ \alpha_{2,2} = \frac{1}{\det_2}\big[U_{2,1} x_{2,2} x_{2,3}(x_{2,2} - x_{2,3}) + U_{2,2} x_{2,1} x_{2,3}(x_{2,1} - x_{2,3}) \\ \qquad + U_{2,3} x_{2,1} x_{2,2}(x_{2,1} - x_{2,2})\big], \end{cases}$$

Once the coefficients are determined, we can compute $g_2 = \frac{\partial P_{\Omega_2}(x_{2,\frac{3}{2}})}{\partial n_1} + \lambda_2 P_{\Omega_2}(x_{2,\frac{3}{2}})$ and

$$\frac{\partial P_{\Omega_1}(x_{1,l-\frac{5}{2}})}{\partial n_1} + \lambda_1 P_{\Omega_1}(x_{1,l-\frac{5}{2}}) = g_2.$$

For simplicity, we consider locally that the point $x_{1,l-\frac{5}{2}} = x_{2,\frac{3}{2}} = 0$. Then, the Robin condition is simplified to: $\alpha_{1,1} + \lambda_1\alpha_{1,2} = \alpha_{2,1} + \lambda_2\alpha_{2,2}$ which is equivalent to

$$\begin{cases} \frac{1}{\det_1}\left[U_{1,l-3}(x_{1,l-1}^2 - x_{1,l-2}^2) + U_{1,l-2}(x_{1,l-3}^2 - x_{1,l-1}^2) + U_{1,l-1}\right. \\ \left.(x_{1,l-2}^2 - x_{1,l-3}^2)\right] + \lambda_1\frac{1}{\det_1}\left[U_{1,l-3}x_{1,l-2}x_{1,l-1}(x_{1,l-2} - x_{1,l-1})\right. \\ \qquad \left. + U_{1,l-2}x_{1,l-3}x_{1,l-1}(x_{1,l-1} - x_{1,l-3})\right. \\ \qquad \left. + U_{1,l-1}x_{1,l-3}x_{1,l-2}(x_{1,l-3} - x_{1,l-2})\right] \\ \\ = \frac{1}{\det_2}\left[U_{2,1}(x_{2,3}^2 - x_{2,2}^2) + U_{2,2}(x_{2,1}^2 - x_{2,3}^2) + U_{2,3}(x_{2,2}^2 - x_{2,1}^2)\right] \\ \quad + \lambda_2\frac{1}{\det_2}\left[U_{2,1}x_{2,2}x_{2,3}(x_{2,2} - x_{2,3}) + U_{2,2}x_{2,1}x_{2,3}(x_{2,1} - x_{2,3})\right. \\ \qquad \left. + U_{2,3}x_{2,1}x_{2,l-2}(x_{2,1} - x_{2,2})\right]. \end{cases}$$

We notice that, for matching meshes the position $x_{1,l-3}$ equals $x_{2,1}$, respectively $x_{1,l-2} = x_{2,2}$ and $x_{1,l-1} = x_{2,3}$. Then $\det_1 = \det_2 = \det$. Before computing the Robin condition, we suppose the closest ghost cells to the domain have already been filled by a Dirichlet condition, thus $U_{2,1} = U_{1,l-3}$ and $U_{1,l-2} = U_{2,2}$. Moreover, for matching meshes and $\lambda_1 = \lambda_2$ we get $U_{1,l-1} = U_{2,3}$ which is a simple Dirichlet condition for one step schemes (Euler Explicit and Runge Kutta). We can treat similarly the Robin condition on Ω_2 and reach the same conclusion.

b) The next idea is to fix the Robin condition no longer on a common interface to both sub-domains, but in the centre of the most exterior ghost cell of each sub-domain. This technique will provide different positions for the Robin condition following the target (see fig.2.12) and more important, the right term of the condition, the one computed by the source will no longer use a value given by a Dirichlet condition.

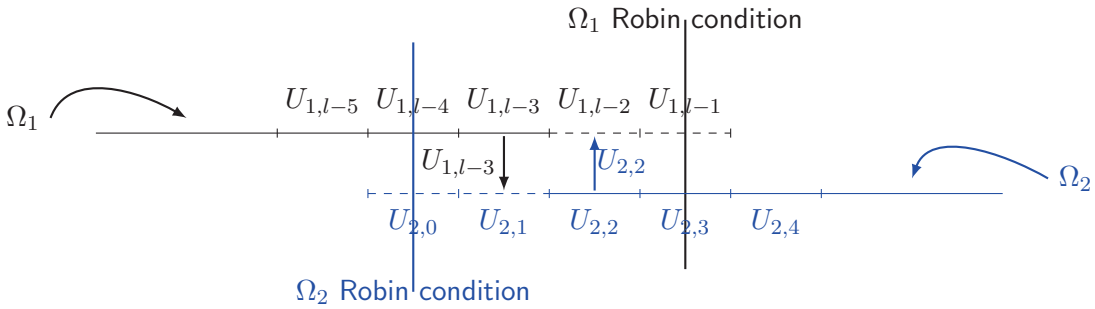


Figure 2.12: Case 2: New position of Robin boundary for two fictitious overlapping sub-domains in the centre of the ghost cells.

The values to fill out are still those of the most exterior ghost cells. When Ω_1 is the target we recall that the value to fill out is $U_{1,l-1}$. Adopting the same technique of polynomial approximation, the first polynomial is identical:

$$P_{\Omega_1}(x) = \alpha_{1,0}x^2 + \alpha_{1,1}x + \alpha_{1,2},$$

with the coefficients described in the previous paragraph. On the source, the expression of the polynomial changes, as the imposed conditions to find the polynomial change. We find

$$P_{\Omega_2}(x) = \alpha_{2,0}x^2 + \alpha_{2,1}x + \alpha_{2,2},$$

such as

$$\begin{cases} P_{\Omega_2}(x_{2,1}) = U_{2,2}, \\ P_{\Omega_2}(x_{2,2}) = U_{2,3}, \\ P_{\Omega_2}(x_{2,3}) = U_{2,4}. \end{cases}$$

The system to solve is:

$$\begin{cases} \alpha_{2,0}x_{2,2}^2 + \alpha_{2,1}x_{2,2} + \alpha_{2,2} = U_{2,2}, \\ \alpha_{2,0}x_{2,3}^2 + \alpha_{2,1}x_{2,3} + \alpha_{2,2} = U_{2,3}, \\ \alpha_{2,0}x_{2,4}^2 + \alpha_{2,1}x_{2,4} + \alpha_{2,2} = U_{2,4}. \end{cases}$$

We note $\det_2 = (x_{2,2} - x_{2,3})(x_{2,3} - x_{2,4})(x_{2,2} - x_{2,4})$ and solve the system of equations to find the coefficients:

$$\begin{cases} \alpha_{2,0} = \frac{1}{\det_2}\big[U_{2,2}(x_{2,3} - x_{2,4}) + U_{2,3}(x_{2,4} - x_{2,2}) + U_{1,4}(x_{2,2} - x_{2,3})\big], \\ \alpha_{2,1} = \frac{1}{\det_2}\big[U_{2,2}(x_{2,4}^2 - x_{2,3}^2) + U_{2,3}(x_{2,2}^2 - x_{2,4}^2) + U_{2,4}(x_{2,3}^2 - x_{2,2}^2)\big], \\ \alpha_{2,2} = \frac{1}{\det_2}\big[U_{2,2}x_{2,3}x_{2,4}(x_{2,3} - x_{2,4}) + U_{2,3}x_{2,2}x_{2,4}(x_{2,2} - x_{2,4}) \\ \qquad + U_{2,4}x_{2,2}x_{2,3}(x_{2,2} - x_{2,3})\big]. \end{cases}$$

The new Robin condition for Ω_1 can be written as:

$$\frac{\partial P_{\Omega_1}(x_{1,l-1})}{\partial n_1} + \lambda_1 P_{\Omega_1}(x_{1,l-1}) = \frac{\partial P_{\Omega_2}(x_{2,3})}{\partial n_1} + \lambda_2 P_{\Omega_2}(x_{2,3}) \overset{noted}{=} g_2.$$

Locally in Ω_1, we consider that the point $x_{1,l-1} = x_{2,3} = 0$ and simplify the Robin condition to $\alpha_{1,1} + \lambda_1\alpha_{1,2} = \alpha_{2,1} + \lambda_2\alpha_{2,2}$ that we develop to

$$\begin{cases} \frac{1}{\det_1}\big[U_{1,l-3}(x_{1,l-1}^2 - x_{1,l-2}^2) + U_{1,l-2}(x_{1,l-3}^2 - x_{1,l-1}^2) + U_{1,l-1} \\ (x_{1,l-2}^2 - x_{1,l-3}^2)\big] + \lambda_1\frac{1}{\det_1}\big[U_{1,l-3}x_{1,l-2}x_{1,l-1}(x_{1,l-2} - x_{1,l-1}) \\ \qquad + U_{1,l-2}x_{1,l-3}x_{1,l-1}(x_{1,l-1} - x_{1,l-3}) \\ \qquad + U_{1,l-1}x_{1,l-3}x_{1,l-2}(x_{1,l-3} - x_{1,l-2})\big] \\ \\ = \frac{1}{\det_2}\big[U_{2,2}(x_{2,4}^2 - x_{2,3}^2) + U_{2,3}(x_{2,2}^2 - x_{2,4}^2) + U_{2,4}(x_{2,3}^2 - x_{2,2}^2)\big] \\ \quad + \lambda_2\frac{1}{\det_2}[U_{2,2}x_{2,3}x_{2,4}(x_{2,3} - x_{2,4}) + U_{2,3}x_{2,2}x_{2,4}(x_{2,2} - x_{2,4}) \\ \qquad + U_{2,4}x_{2,2}x_{2,3}(x_{2,2} - x_{2,3})]. \end{cases}$$

We notice that, for matching meshes the position $x_{1,l-2}$ equals $x_{2,2}$, respectively $x_{1,l-1} = x_{2,3}$ and the equation can be simplified. In the figures 2.15, 2.16, 2.18, 2.20, this condition (b)) is denoted as Dirichlet-Robin.

2.5.3 Robin (Fourier) Boundary Condition

We split this section into two parts. In the first part we continue the previous ideas and work on minimal overlap to strictly fulfil the ghost cells. In the second part we reconsider the possibility of a larger overlap and discuss the possibility of implementing a full Robin type transmission condition instead of a simple Dirichlet type one.

1. Overlap of half stencil size

Another idea is to suppose, as in the case (a) that the Robin boundary condition is fixed on the exact interface between the two sub-domains (represented on fig.2.13).

$$\begin{cases} \frac{\partial U^k_{1,l-\frac{5}{2}}}{\partial n_1} + \lambda_1 U^k_{1,l-\frac{5}{2}} = \frac{\partial U^{k-1}_{2,\frac{3}{2}}}{\partial n_1} + \lambda_2 U^{k-1}_{2,\frac{3}{2}} \overset{noted}{=} g_2^{k-1}, \\ \frac{\partial U^k_{2,\frac{3}{2}}}{\partial n_2} + \lambda_2 U^k_{2,\frac{3}{2}} = \frac{\partial U^{k-1}_{1,l-\frac{5}{2}}}{\partial n_2} + \lambda_1 U^{k-1}_{1,l-\frac{5}{2}} \overset{noted}{=} g_1^{k-1}. \end{cases}$$

The same polynomial technique is considered to approximate the conservative vector:

$$P_\Omega(x) = \alpha_0 x^2 + \alpha_1 x + \alpha_2. \tag{2.7}$$

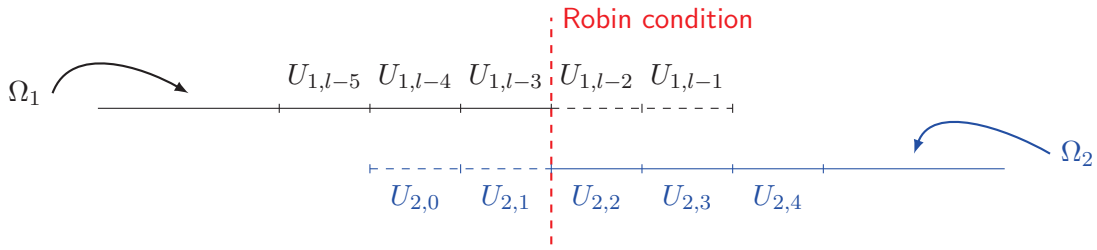


Figure 2.13: Case 3: Position of Robin boundary between two sub-domains. The Robin boundary is imposed as in 2.11, but in this case all ghost cells are filled with values issued from the Robin boundary condition.

This time we propose to impose the Robin condition on the common interface of two sub-domains and to fill out both ghost cells with values issued from this polynomial approximate of the Robin condition. In this case the convective flux and the diffusive flux are computed with values issued from the Robin condition.
On the sub-domain Ω_1, the values to fill out are $U_{1,l-1}$ and $U_{1,l-2}$. In order to find the coefficients we impose that:

$$\begin{cases} P_{\Omega_1}(x_{1,l-3}) = U_{1,l-4}, \\ P_{\Omega_1}(x_{1,l-2}) = U_{1,l-3}, \\ \frac{\partial P_{\Omega_1}(x_{1,l-\frac{5}{2}})}{\partial n_1} + \lambda_1 P_{\Omega_1}(x_{1,l-\frac{5}{2}}) = g_2. \end{cases}$$

where we note $g_2 = \frac{\partial P_{\Omega_2}(x_{2,\frac{5}{2}})}{\partial n_1} + \lambda_2 P_{\Omega_2}(x_{2,\frac{5}{2}})$. The system to solve is equivalent to a system of three equations with three unknowns $\alpha_{1,0}, \alpha_{1,1}$ and $\alpha_{1,2}$:

$$\begin{cases} \alpha_{1,0} x_{1,l-4}^2 + \alpha_{1,1} x_{1,l-4} + \alpha_{1,2} = U_{1,l-4}, \\ \alpha_{1,0} x_{1,l-3}^2 + \alpha_{1,1} x_{1,l-3} + \alpha_{1,2} = U_{1,l-3}, \\ \alpha_{1,0}(x_{1,l-\frac{5}{2}}^2 + 2n_1 x_{1,l-\frac{5}{2}}) + \alpha_{1,1}(\lambda_1 x_{1,l-\frac{5}{2}} + n_1) + \alpha_{1,2}\lambda_1 = g_2. \end{cases}$$

In order to simplify the calculations and without any lost of precision we can locally suppose that $x_{1,l-\frac{5}{2}}$ is the origin of the axis. Then the system is simplified to

$$\begin{cases} \alpha_{1,0} x_{1,l-4}^2 + \alpha_{1,1} x_{1,l-4} + \alpha_{1,2} = U_{1,l-4}, \\ \alpha_{1,0} x_{1,l-3}^2 + \alpha_{1,1} x_{1,l-3} + \alpha_{1,2} = U_{1,l-3}, \\ \alpha_{1,1} n_1 + \alpha_{1,2}\lambda_1 = g_2. \end{cases}$$

We note $\det_1 = (x_{1,l-3} - x_{1,l-4})\left[n_1(x_{1,l-3} + x_{1,l-4}) - x_{1,l-3}x_{1,l-4}\lambda_1\right]$, we solve the system by a Cramer method and obtain:

$$\begin{cases} \alpha_{1,0} = \frac{1}{\det_1}\left[g_2(x_{1,l-4} - x_{1,l-3}) + U_{1,l-3}(n_1 - \lambda_1 x_{1,l-4}) + U_{1,l-4}(\lambda_1 x_{1,l-3} - n_1)\right], \\ \alpha_{1,1} = \frac{1}{\det_1}\left[g_2(x_{1,l-3}^2 - x_{1,l-4}^2) + U_{1,l-3}\lambda_1 x_{1,l-4}^2 - U_{1,l-4}\lambda_1 x_{1,l-3}^2\right], \\ \alpha_{1,2} = \frac{1}{\det_1}\left[g_2 x_{1,l-3}x_{1,l-4}(x_{1,l-4} - x_{1,l-3}) - U_{1,l-3}n_1 x_{1,l-4}^2 + U_{1,l-4}n_1 x_{1,l-3}^2\right]. \end{cases}$$

On the source Ω_2 we impose:

$$\begin{cases} \frac{\partial P_{\Omega_2}(x_{2,\frac{3}{2}})}{\partial n_2} + \lambda_2 P_{\Omega_2}(x_{2,\frac{3}{2}}) = g_1, \\ P_{\Omega_2}(x_{3,2}) = U_{2,2}, \\ P_{\Omega_2}(x_{2,3}) = U_{2,3}, \end{cases}$$

where we note $g_1 = \frac{\partial P_{\Omega_1}(x_{1,l-\frac{5}{2}})}{\partial n_2} + \lambda_1 P_{\Omega_1}(x_{1,l-\frac{5}{2}})$. The system is equivalent to:

$$\begin{cases} \alpha_{2,0}(x^2_{2,\frac{3}{2}} + 2n_2 x_{2,\frac{3}{2}}) + \alpha_{2,1}(\lambda x_{2,\frac{3}{2}} + n_2) + \alpha_{2,2}\lambda_2 = U_{2,1}, \\ \alpha_{2,0}x^2_{2,2} + \alpha_{2,1}x_{2,2} + \alpha_{2,2} = U_{2,2}, \\ \alpha_{2,0}x^2_{2,3} + \alpha_{2,1}x_{2,3} + \alpha_{2,2} = U_{2,3}, \end{cases}$$

We note $\det_2 = (x_{2,3} - x_{2,2})\,(n_2(x_{2,3} + x_{2,2}) - x_{2,2}x_{2,3}\lambda_2)$ and we solve the system using the same assumptions and obtain the following coefficients:

$$\begin{cases} \alpha_{2,0} = \frac{1}{\det_2}\left[g_1(x_{2,2} - x_{2,3}) + U_{2,2}(\lambda_2 x_{1,3} - n_2) + U_{2,3}(n_1 - \lambda_2 x_{2,2})\right], \\ \alpha_{2,1} = \frac{1}{\det_2}\left[g_1(x^2_{2,3} - x^2_{2,2}) - U_{2,2}\lambda_2 x^2_{2,3} + U_{2,3}\lambda_2 x^2_{2,2}\right], \\ \alpha_{2,2} = \frac{1}{\det_2}\left[g_1 x_{2,2}x_{2,3}(x_{2,2} - x_{2,3}) + U_{2,2}n_2 x^2_{2,3} - U_{2,3}n_2 x^2_{2,2}\right]/ \end{cases}$$

We suppose that the unknowns ghost cells belong to the same polynomial as the Robin condition and determine the needed values to compute both the convective and diffusive fluxes. As it is numerically shown below (fig.2.17) this positioning of Robin condition does not work. An alternative is to impose a first Robin condition on the exact interface and a second Robin boundary condition between the two ghost cells with the same polynomial approximate. In the figures 2.15, 2.16, 2.18, 2.20, this condition b) is denoted as Robin-Robin.

2. Overlap of size larger than half stencil size

Our overall parallel strategy is based on using domain decomposition methods adapted to overlapping sub-domains, with minimal overlap of half stencil size. The cells that overlap exist only to store boundary conditions and are never considered as part of the final solution. Some researchers would even consider the use of ghost cells as a developer choice, independent of the numerical method. For example, additional vectors could replace the use of ghost cells.

It is known that the larger the overlap the faster the convergence is attained. In our previous discussions the Robin boundary condition is computed using an upwind or a downwind scheme. To improve the accuracy, a minimal overlapping domains (not taking into account fictitious cells) will allow the use of a centred scheme.

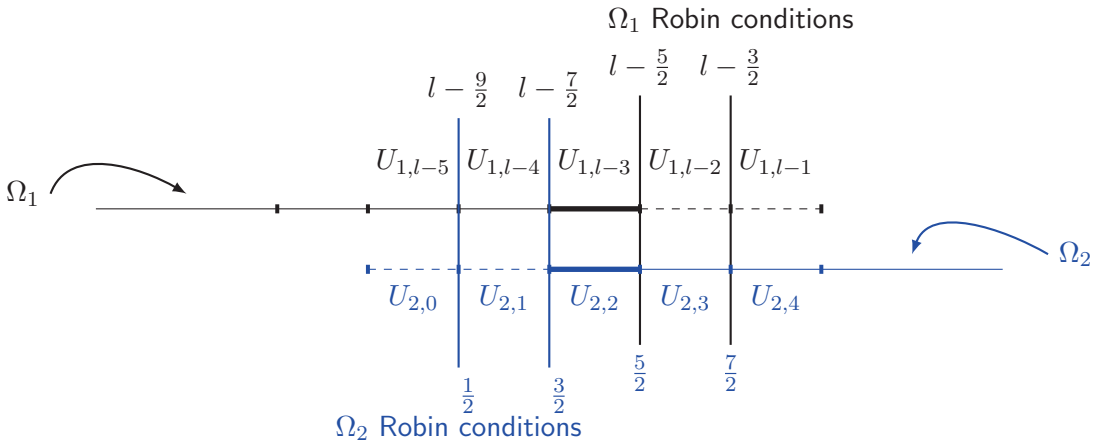


Figure 2.14: Case 4: Robin boundary conditions for two domains. Each cell ghost is filled out from a different Robin boundary condition.

Then, the resulting system is well posed and have two distinct boundary conditions, as on

fig.2.14.

$$\text{On } \Omega_1\text{:}\begin{cases} \dfrac{\partial U^k_{1,l-\frac{5}{2}}}{\partial n_1} + \lambda_1 U^k_{1,l-\frac{5}{2}} = \dfrac{\partial U^{k-1}_{2,\frac{5}{2}}}{\partial n_1} + \lambda_2 U^{k-1}_{2,\frac{5}{2}} \overset{\text{noted}}{=} g^k_{2,\frac{5}{2}}, \\ \dfrac{\partial U^k_{1,l-\frac{3}{2}}}{\partial n_1} + \lambda_1 U^k_{1,l-\frac{3}{2}} = \dfrac{\partial U^{k-1}_{2,\frac{7}{2}}}{\partial n_1} + \lambda_2 U^{k-1}_{2,\frac{7}{2}} \overset{\text{noted}}{=} g^k_{2,\frac{7}{2}}, \end{cases}$$

$$\text{On } \Omega_2\text{:}\begin{cases} \dfrac{\partial U^k_{2,\frac{1}{2}}}{\partial n_2} + \lambda_2 U^k_{2,\frac{1}{2}} = \dfrac{\partial U^{k-1}_{1,l-\frac{9}{2}}}{\partial n_2} + \lambda_1 U^{k-1}_{1,l-\frac{9}{2}} \overset{\text{noted}}{=} g^k_{1,l-\frac{9}{2}}, \\ \dfrac{\partial U^k_{2,\frac{3}{2}}}{\partial n_2} + \lambda_2 U^k_{2,\frac{3}{2}} = \dfrac{\partial U^{k-1}_{1,l-\frac{7}{2}}}{\partial n_2} + \lambda_1 U^{k-1}_{1,l-\frac{7}{2}} \overset{\text{noted}}{=} g^k_{1,l-\frac{7}{2}}. \end{cases}$$

The gradients can be approximated using a simple centred differencing scheme.

$$\text{On } \Omega_1\text{:}\begin{cases} \dfrac{U^k_{1,l-2} - U^k_{1,l-3}}{x_{l-2} - x_{l-3}} + \lambda_1 \dfrac{U^k_{1,l-2} + U^k_{1,l-3}}{2} = g^k_{2,\frac{5}{2}}, \\ \dfrac{U^k_{1,l-1} - U^k_{1,l-2}}{x_{l-1} - x_{l-2}} + \lambda_1 \dfrac{U^k_{1,l-1} + U^k_{1,l-2}}{2} = g^k_{2,\frac{7}{2}}, \end{cases}$$

$$\text{On } \Omega_2\text{:}\begin{cases} \dfrac{U^k_{2,0} - U^k_{2,1}}{x_1 - x_0} + \lambda_2 \dfrac{U^k_{2,1} + U^k_{2,0}}{2} = g^k_{1,l-\frac{9}{2}}, \\ \dfrac{U^k_{2,1} - U^k_{2,2}}{x_2 - x_1} + \lambda_2 \dfrac{U^k_{2,2} + U^k_{2,1}}{2} = g^k_{1,l-\frac{7}{2}}. \end{cases}$$

In two sequential steps all unknowns are found, firstly, the most interior ones $U^k_{1,l-2}$ and $U^k_{2,1}$ and secondly $U^k_{1,l-1}$ and $U^k_{2,0}$.

$$\text{On } \Omega_1\text{:}\begin{cases} U^k_{1,l-2} = \dfrac{1}{(0.5\lambda_1(x_{l-2} - x_{l-3}) + 1)}\Big[(x_{l-2} - x_{l-3}) g^k_{2,\frac{5}{2}} - \big(0.5\lambda_1(x_{l-2} - x_{l-3}) - 1\big) U^k_{1,l-3}\Big], \\ U^k_{1,l-1} = \dfrac{1}{(0.5\lambda_1(x_{l-1} - x_{l-2}) + 1)}\Big[(x_{l-1} - x_{l-2}) g^k_{2,\frac{7}{2}} - \big(0.5\lambda_1(x_{l-1} - x_{l-2}) - 1\big) U^k_{1,l-2}\Big], \end{cases}$$

$$\text{On } \Omega_2\text{:}\begin{cases} U^k_{2,1} = \dfrac{1}{(0.5\lambda_2(x_2 - x_1) + 1)}\Big[(x_2 - x_1) g^k_{1,l-\frac{7}{2}} - \big(0.5\lambda_2(x_2 - x_1) + 1\big) U^k_{2,2}\Big], \\ U^k_{2,0} = \dfrac{1}{(0.5\lambda_2(x_1 - x_0) + 1)}\Big[(x_1 - x_0) g^k_{1,l-\frac{9}{2}} - \big(0.5\lambda_2(x_1 - x_0) + 1\big) U^k_{2,1}\Big]. \end{cases}$$

The Dirichlet boundary conditions apply naturally to any size of the overlap. In what concerns the mixed type boundary condition, we observe that only the second proposition (Robin condition imposed in the centre of the ghost cells) can adapt to overlap larger than half stencil size. In the figures 2.15, 2.16, 2.18, 2.20, this condition (b)) is denoted as Robin-Robin.

2.6 Convergence and Stopping criteria

Consider as example the resolution of the following system of equations (similar to the main operation in an implicit Euler scheme).

$$\begin{cases} \eta u - \partial_x^2 u = f \text{ in } \Omega = [a, b], \\ u = g \text{ on } \{a, b\}, \end{cases}$$

To study the efficiency of the presented transmission conditions, we split Ω in two subdomains Ω_1 and Ω_2, and solve the resulting system with a parallel Schwarz method. For Dirichlet boundary type transmission condition the parallel Schwarz algorithm is the following:

1. For k=,1,2,.., the Schwarz iteration index solve:

$$\begin{cases} \eta u_1^k - \partial_x^2 u_1^k = 0 & \text{in } \Omega_1, \\ u_1^k = g & \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ u_1^k = u_2^{k-1} & \text{on } \Gamma_1, \end{cases} \qquad \begin{cases} \eta u_2^k - \partial_x^2 u_2^k = 0 & \text{in } \Omega_2, \\ u_2^k = g & \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ u_2^k = u_1^{k-1} & \text{on } \Gamma_2. \end{cases}$$

We solve the system of equations with a Finite Volume method, and use the same centred numerical scheme as for the viscous fluxes in the Navier–Stokes system of equations, 1.2.4. The time is discretised by a first order Euler Explicit scheme (see section 1.3.1). We have studied the convergence rate for different type of boundary conditions. For all presented results, η equal 1, the domain of definition is $\Omega = [0, 1]$, the initial solution is zero. We impose an inflow boundary condition of $g = 2$ in $\{a\}$ and transmissible boundary condition in $\{b\}$. For the numerical simulation we have imposed a tolerance inside the Schwarz process of 10^{-10}. Inside one subdomain the solution is not converged, we stop at two iterations. The reasons is that, we are interest in implicit schemes, with inexact solvers which are stopped before convergence. Based on the previous cases two scenarios are possible: decomposition with overlap of stencil size and decomposition with overlap of larger size than the stencil size. We define $\Omega_1 = [a, c + \theta\Delta x]$ and $\Omega_2 = [c - \theta\Delta x, b]$, $\Gamma_1 = \{c + (\theta - 1)\Delta x, c + \theta\Delta x\}$ and $\Gamma_2 = \{c - \theta\Delta x, c - (\theta - 1)\Delta x\}$ where θ denotes the overlap size. In the first case we pose $\theta = 2$, the size of half the stencils, and in the second case $\theta = 3$.

Overlap of stencil size

We distinguish three main types of transmission boundary conditions:

1. Dirichlet-Dirichlet: Dirichlet transmission condition (section 2.5.1) to compute both convective and diffusive fluxes;
2. Dirichlet-Robin: Mixed Dirichlet/Robin transmission condition (section 2.5.2.b) to compute the fluxes. The Robin condition is imposed in the centre of the more external ghost cells;
3. Robin-Robin: Robin transmission condition (section 2.5.3.2) to compute both convective and diffusive fluxes.

On fig.2.15 we can see the resulting convergence rate for the above transmission conditions.

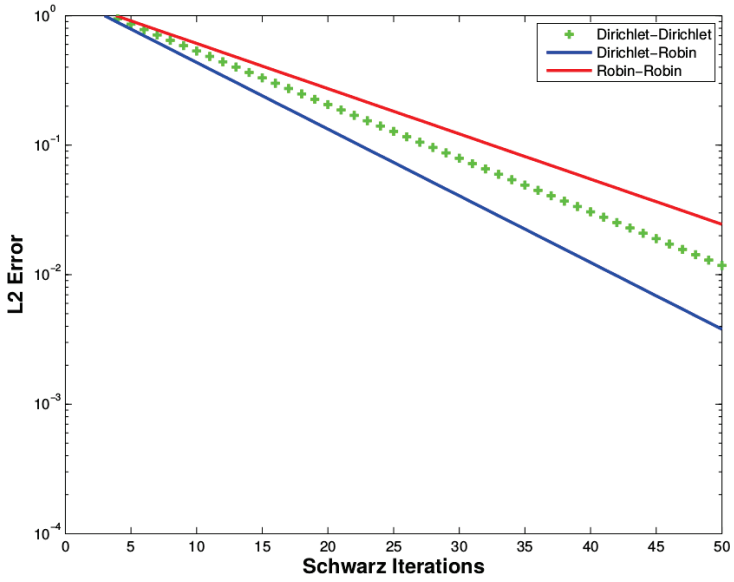


Figure 2.15: Convergence comparison for non-matching meshes. The L2 error stands for L2 error norm between the computed solution and the exact solution .

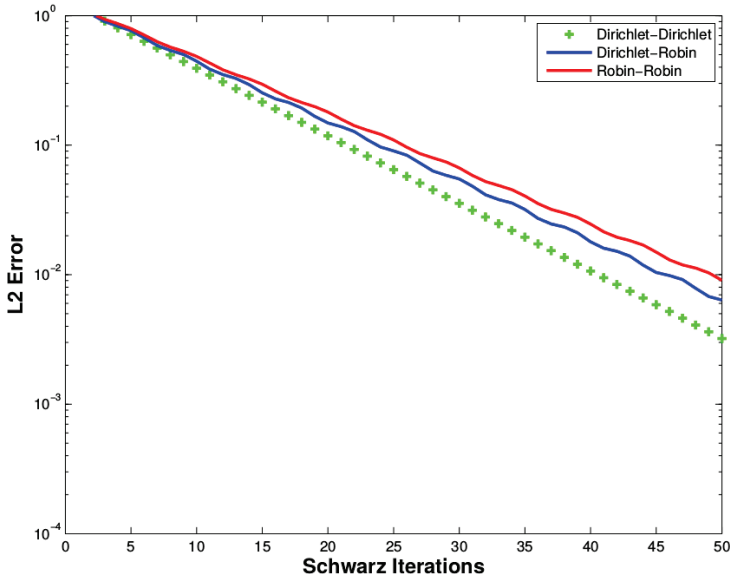


Figure 2.16: Convergence comparison. Dirichlet-Robin stands for both possibilities of imposing the Robin boundary since they give identical results. Contrary as expected, for non-matching meshes the solver with Dirichlet-Dirichlet boundary condition converges faster.

For matching cells, the mixed transmission condition improves the convergence rate and diminishes the necessary number of Schwarz processes. But it is no longer the case for non-matching meshes and our interest focused mainly on non-matching meshes, which adapt better to applications and to SWR methods. On fig.2.16 we see that the Dirichlet-Dirichlet transmission condition is faster than the two others. The Robin-Robin transmission boundary condition is expected to improve the convergence, but it is actually the worst case. We believe that it is due to polynomial approximations. Moreover, this result stands for any Robin coefficient.

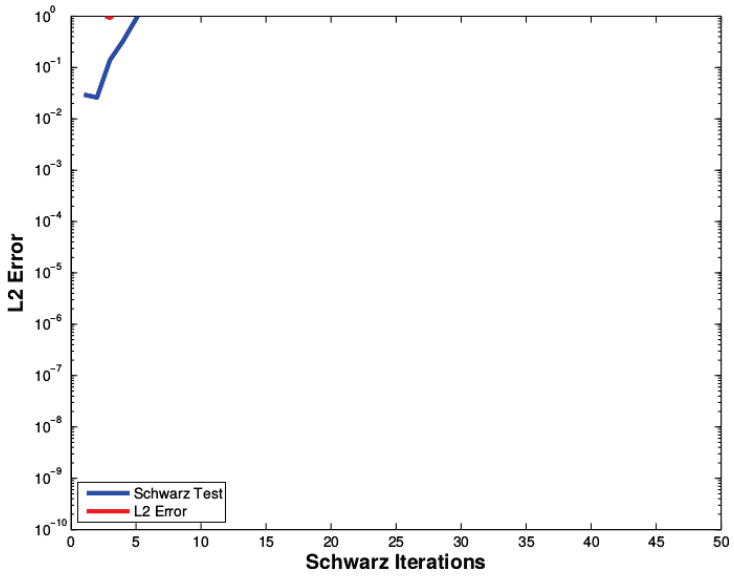


Figure 2.17: Divergence results for two boundaries values arising of a Robin condition imposed at the interface.

Overlap of larger size than the stencil size

In the second scenario we distinguish the following possibilities to impose the transmissive conditions:

1. Dirichlet-Dirichlet: Dirichlet transmission condition (section 2.5.1) to compute both convective and diffusive fluxes;
2. Dirichlet-Robin: Mixed Dirichlet/Robin transmission condition (section 2.5.2) to compute the fluxes. The Robin condition is imposed in the centre of the most external ghost cells. It seems impossible to impose a Robin condition in a common interface, especially for non-matching meshes;
3. Robin-Robin: Robin transmission condition (section 2.5.3.2) to compute both convective and diffusive fluxes.

On fig.2.18 and fig.2.20 we present the convergence comparisons for all three types of transmission conditions for non-matching meshes. These tests confirm that an increased overlap improves considerably the convergence rate and diminishes the necessary number of Schwarz precesses for all types of transmission conditions. Moreover, the Robin-Robin transmission condition is the more efficient one. The Dirichlet-Robin transmission condition gives similar results to ones found with the Dirichlet-Dirichlet transmission condition. On fig.2.19 we can see the optimal Robin coefficient for an overlap equal to 3 cells. This coefficient has been found at the end of the Schwarz process and is equal to $\alpha\frac{1}{\sqrt{\min(\Delta x_1,\Delta x_2)}}$ where Δx_1 and Δx_2 are the local space steps. In this case the Robin coefficient equals: $1.05 * \frac{1}{\sqrt{0.03333}} = 5.75$.

On fig.2.21, fig.2.22 and fig.2.23 we compare results found with converged solutions in each sub-domain. We suppose that the local solution is converged is a relative error becomes smaller than 10^{-10}. For non-matching meshes and overlap equal to half the stencil size ($\theta = 2$), on fig.2.21, the solution found with Dirichlet-Robin transmission conditions converges faster then the solution found with Dirichlet-Dirichlet or Robin-Robin transmission conditions. The faster convergence is obtained when the values of the ghost cells are given by one Robin transmission condition imposed at the interface. In this last situation, for non-converged local solutions (insufficient stopping criteria), the global solution diverges. For non-matching meshes and

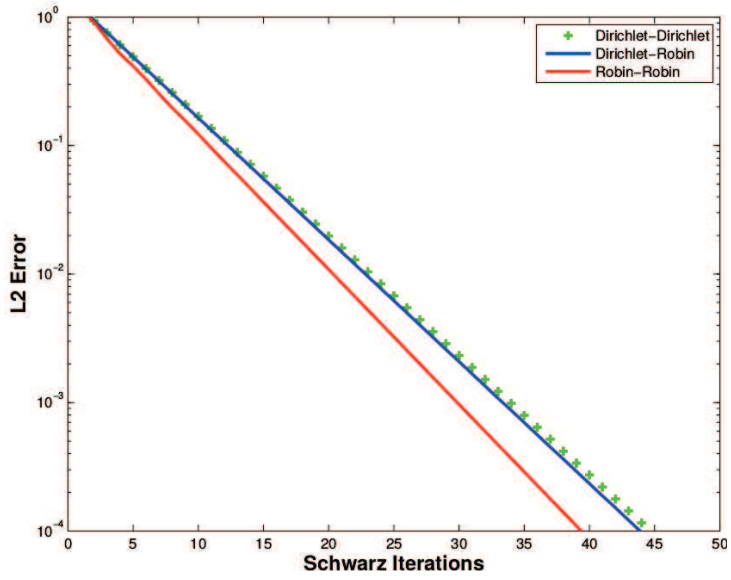


Figure 2.18: Convergence comparison for overlap $\theta = 3$ cells, within the overlap, two cells are used to impose the transmission conditions. The results have been found with an optimal Robin coefficient.

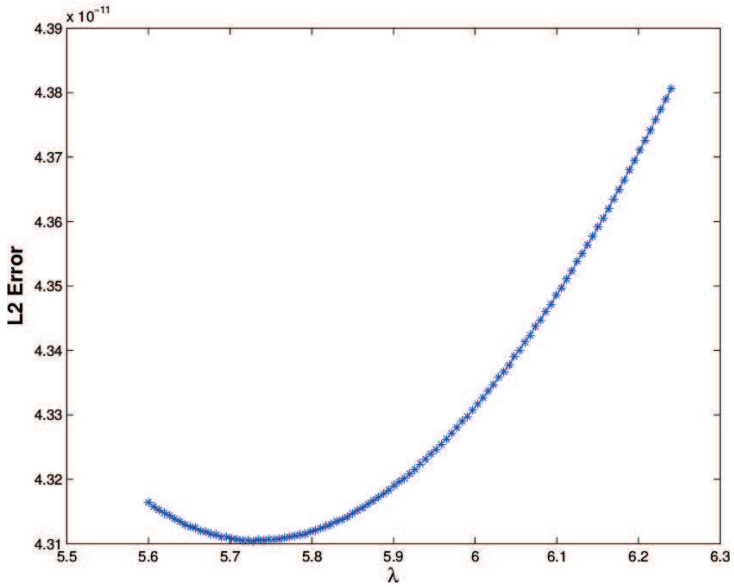


Figure 2.19: Research for the optimal Robin coefficient for an overlap equal to 3 cells. The research has been done for converged solution inside the Schwarz process.

overlap $\theta = 3$ and $\theta = 4$ cells, the solution found with Robin-Robin condition converges faster. The cost to find a converged local solution is high and does not match our strategy for implicit methods.

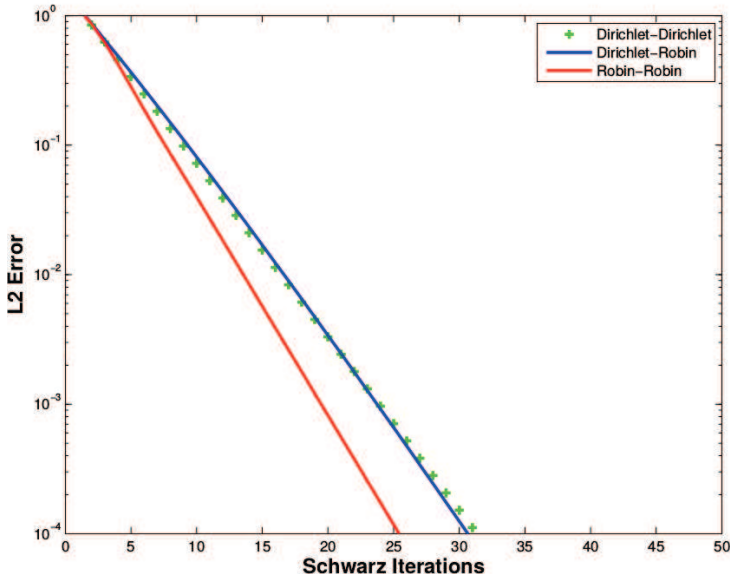


Figure 2.20: Convergence comparison for overlap $\theta = 4$ cells, within the overlap, two cells are used to impose the transmission conditions. The results have been found with an optimal Robin coefficient.

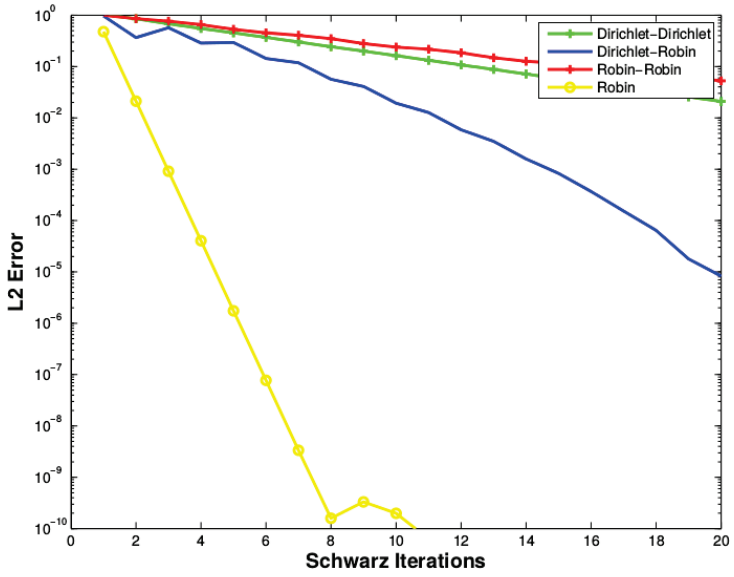


Figure 2.21: Convergence comparison for overlap $\theta = 2$ cells and non-matching meshes. We call Robin the solution found with a Robin condition imposed at the interface. The results have been found with an optimal Robin coefficient.

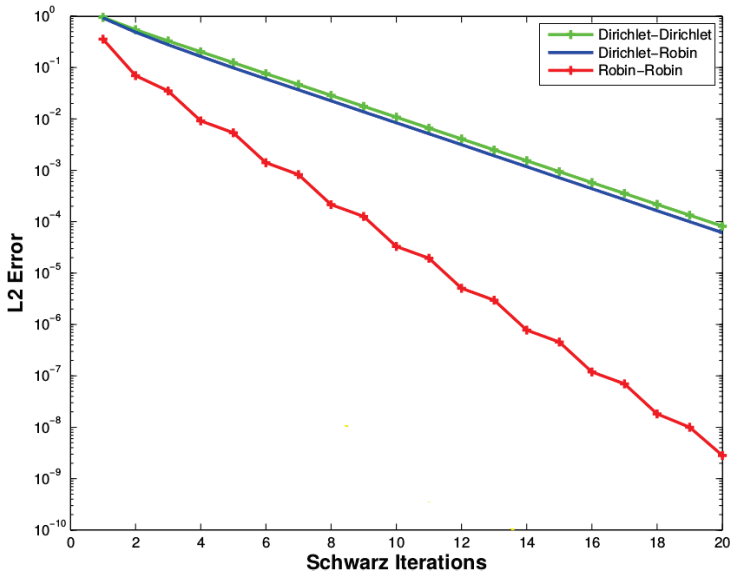


Figure 2.22: Convergence comparison for overlap $\theta = 3$ cells and non-matching meshes. The results have been found with an optimal Robin coefficient.

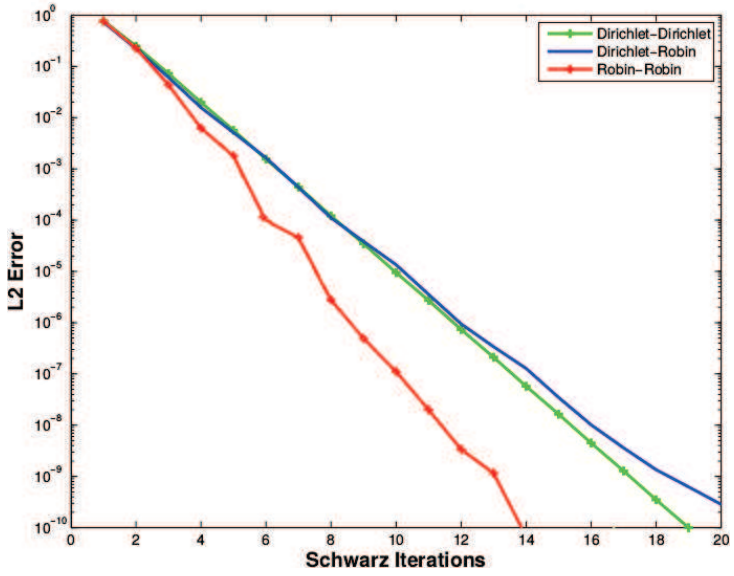


Figure 2.23: Convergence comparison for overlap $\theta = 4$ cells and non-matching meshes. The results have been found with an optimal Robin coefficient.

What about stopping criteria?

In what concerns the Navier–Stokes solver, several convergence criteria must be set. Issues can arise from non-linear robustness (discretisation order, initial time step), choice of parameters (convergence tolerance), Schwarz parameters (sub-domain number, overlap size, stopping tolerance), from local and global strategy. What are the relative tolerances that determine the convergence criteria for the problem? The choice of these variables must be done very carefully. It is well known that the stopping criteria is very important as it may lead to inaccurate solutions

when the iteration is stopped too soon and also to extra costs when a problem is over-solved. In our Navier–Stokes computations, stopping criteria are based on residual norms, while other authors use a stopping criteria based on error norms (as in the case of the Laplacian). Moreover, several stopping criteria are combined in order to ensure convergence, but also to avoid infinite loops due to numerical errors. The error norm is usually more accurate, but requires the knowledge of the exact solution, which is often not known (still as in the Laplacian case). Different analyses based on numerical experiments have been made, especially to determine the best stopping criteria for iterative solvers (for example, see [19]).

When can we say that the Schwarz process has converged? Since we are using the overlap (fictitious or not), the first natural way to check the continuity of a global solution over more than one domain is to check if the overlap region is the same, with a chosen tolerance to the computed solution. This criteria seems efficient, it is reliable especially when Dirichlet interface conditions are imposed. Actually, using a Dirichlet interface condition we are checking that the values on the overlap region of one sub-domain at time t^n are becoming identical to the values in the overlap region, on the same sub-domain, but at time t^{n+1} which is a classical convergence criteria. In the different cases of Robin interface condition, this criteria may be appropriate or not, depending on the case (see section 2.5). But under this form, the computation of this criteria is costly, since, at each iteration some communication between sub-domains is needed to establish the leap between this overlap regions. Also, the tasks on each processor must be finished before computing this variation.

Based on this same idea, we propose, as common stopping criteria the relative error (L^∞) between the actual solution and the previous one on the most exterior rows and columns of each computational sub-domain (red on fig.2.24).

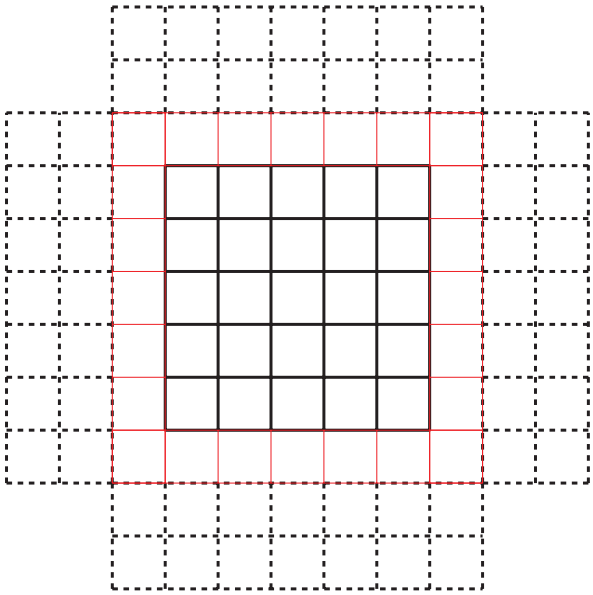


Figure 2.24: 2D Cartesian mesh. One can see a 2D computational domain Ω of size 7×7 cells and the fictitious overlap of stencil size two. In red we highlight the cells involved in the convergence stopping criteria.

We suppose that we have achieved convergence when the solution on the internal boundary of one domain ceases to change. In addition we also stop the iterations when the difference between two relative errors is identical. We suppose then that the boundary condition cannot be improved any more so the solution can not be improved for higher number of iterations.

On fig.2.25 and fig.2.26 we have plotted the convergence rate found to solve the previous 1D problem for non-matching meshes and overlap equal to three space cells. We change the stopping

criteria in the Schwarz process. We compare the L2 error norm (not always available) with our proposed convergence test that checks the time evolution of the cells around the computational domain (without the ghost cells). The proposed criteria is as efficient as the L2 error norm.

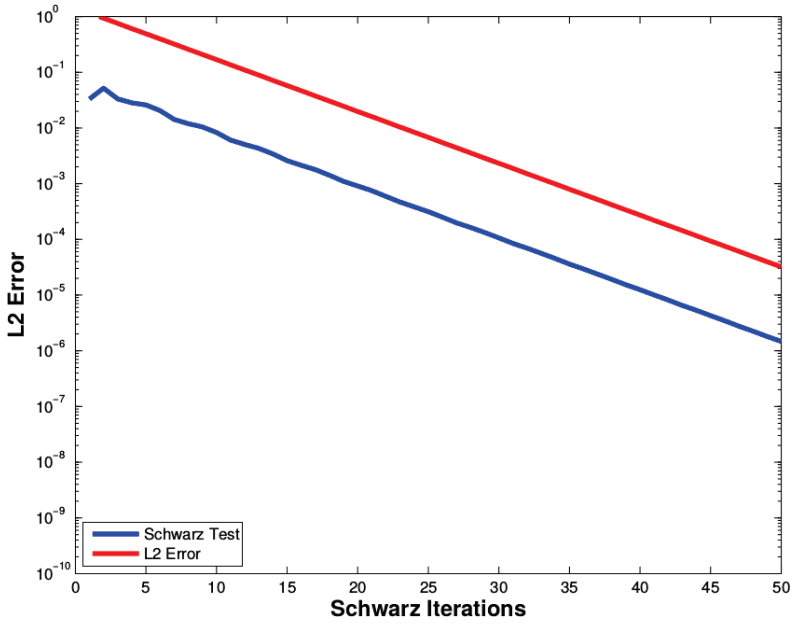


Figure 2.25: Dirichlet-Dirichlet convergence rate with the L2 error norm and the new stopping criteria: L2 variation in time of the first and last cells of the domain without ghost cells.

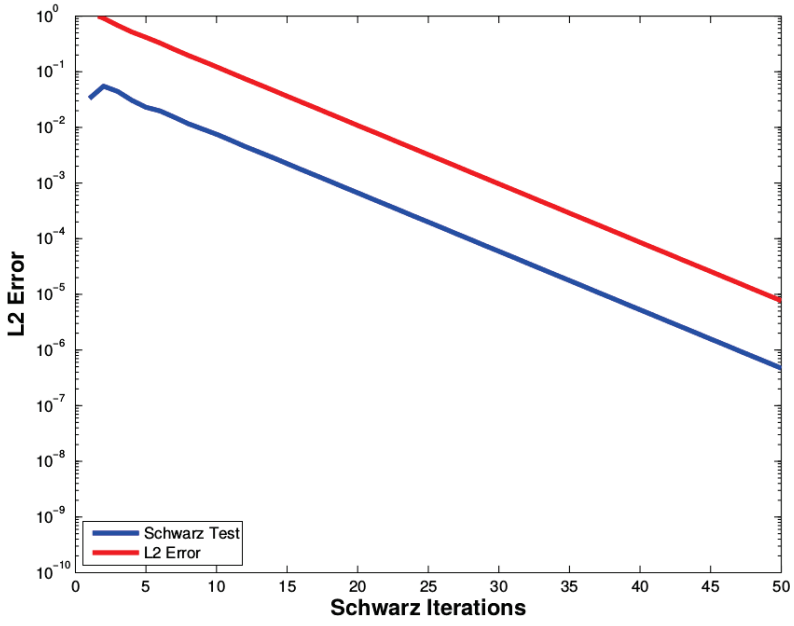


Figure 2.26: Robin-Robin convergence rate with the L2 error norm and the new stopping criteria: L2 variation in time of the first and last cells of the domain without ghost cells.

Chapter 3

CFD Code organisation and description

Contents

Within Onera, missions such as research, construction and operation of the associated experimental facilities, technical analysis in the aeronautic field are department distributed and correlated. Designing and manufacturing civil aircraft, military aircraft, helicopters, propulsion systems, orbital systems, space transport, missile systems, defense systems, networked systems and security systems are goals attained by the Onera researchers. Nevertheless, computation codes, methods and tools are developed and constantly improved. We shortly present some of the most pertinent Onera software for which our work is of use:

- **CEDRE** (*http://cedre.onera.fr*) is a multi-physics, multi-mesh, multi-model and multi-method platform in the fields of energetics and propulsion developed since 90's;
- **Zebulon**/Z-set (*http://www.zset-software.com*) is a finite element code for material - oriented analysis software developed in collaboration with other academic partners (École des Mines ParisTech (France), NW Numerics & Modelling, Inc (USA),...);
- **elsA** (*http://elsa.onera.fr*) is a platform for complex CFD simulations. The acro-nym stands for "ensemble logiciel de simulation en Aérodynamique" and can be translated as "software for Aerodynamic simulations";

The elsA project ([32]) was started in 1997 by the Computational Fluid Dynamics and Aeroacoustics (DSNA) department. DSNA is still the main contributor in the constant evolution and extension of the code, but nowadays other Onera departments constantly contribute to its evolution: DAAP (Applied Aerodynamics), DMAE (Aerodynamics and Energetics Modelling), DAFE (Fundamental and Experimental Aerodynamics), DEFA (Fundamental and Applied Energetics), DMPH (Physics and Instrumentation). elsA's object-oriented CFD library includes a rich environment for modelling multi-disciplinary aerodynamic applications, complex external and internal flow simulations. It deals with large scale problems, solves the compressible Navier–Stokes equations in a cell centred finite volume formulation, uses turbulence models. To simulate the fluid around a helicopter model in a wind tunnel, most code uses mesh reconstruction, but the Onera elsA code uses also a special technique, called the Chimera method that allows meshes to move with the body. The platform uses the Python language as user interface language, C++ and Fortran languages to translate object concepts and implement methods.

Let us recall that the main objective of our work is to prove the efficiency of new, improved parallelisation techniques over the one already used in elsA (defined as Newton-Partitioning Algorithm in the previous chapter), to build them in a modular form that allows a simple use and integration within elsA or any other platform. The presented domain decomposition methods are validated on simple cases and with no turbulence model. We expect good performance when turbulence models and unstructured grids are used. The elsA industrial code should be able to use the new modular code only through its Python interface and with no major modification.

In this chapter we discuss the means of parallel computing, how to understand, organize and tune the parallel domain decomposition method, how to carefully exploit the computer's architecture and make sure that the algorithm is flexible and portable. We present the structure and the main modules of the implemented code and briefly present the tools (OpenMP, MPI and GPU CUDA) that enable the parallel computation, their evolution, advantages and limits. We go from basic to advanced notions of computer hierarchy, discuss new functionalities and their management. We describe techniques of efficiently parallel programming by message passing (MPI), loops parallelisation (OpenMP) for numerical simulations. We introduce and aim to render the GPU architecture comprehensible to the reader so he can appreciate its utility. Example of applications are shown for different computer architecture. In the last section we discuss ways of measuring parallel efficiency.

3.1 Code description and programming techniques

Our new code, like the elsA code, is built to satisfy software imperative qualities: efficiency, usability (user-friendly), interoperability, flexibility (fast and easy integration of other numerical methods), scalability (works for any volume of data), modularity (made of independent units and modules that can be tested and modified separately), reusability (used for different tests or purpose with only minor modifications) and economy (minimize the cost to build and solve).

To have a modular program driven by reusability, the Python language was chosen. Python is an object oriented and structured programming language built to be accessible and highly readable by users. It is a high-level language, allowing scripting language similar to Matlab and it is flexible, with only few constraints (unlike strongly typed languages, such as C++). Available on all types of platform (Unix, Windows and Mac OS), Python disposes of a wide range of data structures (list, dictionaries, tuples, etc.), scientific and graphical libraries (numpy, scipy, CGNS, mpi, vtk, matplotlib, etc.). It supports Fortran (f2py), C/C++ (API C, Cython, etc.). Python is well adapted to large projects and seems to be ideal for our purpose, but it has a major drawback: it is very slow, at least 30 times slower than C or Fortran. Therefore, it is used as an interface and not to compute large data systems of equations. To do so, we use a tool capable of translating Python scripts in C or making direct level API calls in the Python interpreter, called Cython. Cython, (extension .pyx) is a derivative of the pyrex language, it allows declaring C variables and calling C functions. It allows the compiler to generate C codes thus speeding up the computation. The resulting code performance is comparable with the one obtained with Fortran, C or C++. We use the C language (highly portable) for sections with high CPU consumption (for loops over the entire mesh cells or interfaces), as it is a compiled language that creates fast and efficient executable files and allows control over the memory (dynamic memory allocations, recursive functions, pointers, etc.).

Each module can contain one or more C/Cython files. Let us take for example the module Boundaries. Inside this module we can find descriptions of different boundary conditions. Let us detail the files concerning the inflow boundary condition:

- **Factory.pyx** is a cython file which exists within each module, that extracts initial data directly from the initial CGNS tree, whose detailed description is given on the next page, and creates instances related to the CGNS tree description;
- **CInflowBoundary.c** is a C file containing the memory allocation needed to compute the inflow boundary, updates of the boundary and deallocation of the temporary variables and/or vectors of variables that are not needed any-more;
- **CInflowBoundary.h** is a C header file which contains the links to external or internal libraries, the definition of the InflowBoundary structure (example: typedef struct{ Boundary* pt_base; double* initValues;} InflowBoundary;) and the prototypes of C functions built in CInflowBoundary.c;
- **CInflowBoundary.pxd** contains prototypes of C functions that are shared with other modules via cython. In other words it is used to render visible C functions in cython. We note that not all C functions must be visible, some may only be used by other functions locally declared and used in C;
- **InflowBoundary.pyx** is the cython file that will directly communicate with other modules, it contains the definition of the InflowBoundary class with directives to C functions;

CGNS Data Tree

In order to facilitate the manipulation of initial numerical data we use the CFD General Notation System, called CGNS. This means that we transform and store all the information into a data structure. CGNS is a notation introduced by NASA and Boeing in the aim to standardise the CFD initial data and results, to facilitate the exchange across computed platform, between sites and applications and to help building an aerodynamic archive. CGNS is based on HDF5 (Hierarchical Data Format) technology, a data model, library and file format that stores and manages hierarchical data (XML documents, in-house data formats). HDF5 deals with massive and complex data and supports parallel I/O (input/output) access. Moreover, visualisation tools, such as tecplot, recognise CGNS data format. Thus, we adopted this idea and used similar conventions for the storage of input and output data.

A CGNS tree is a collection of nodes, each node can contain references to other nodes called children. A node is built as a Python sequence (list) that contains four entries: *name*, *value*, *node* or *child* and *type*.

Name : string,
Value : numpy array,
Node : list of nodes,
Type : string.

Example:

- tree = ['ShockTube', timeOut, [Subdomain1, Subdomain2], 'CGNSBase_t'],
- Subdomain1 = ['Tube', numpy.array([[6,1,1],[5,1,1]]), [Grid, Boundaries, InitSol, GridConnectivity], 'Subdomain_t'].

The field *name* of a node is a string that contains a key word to describe the simulation, usually the name of the computational test. The field *value* is a N-dimensional array that can contain different initial data (for example the origins of the computational domain, the size of a grid via the number of vertices or cells, initial values on boundaries, interface size and coordinates, initial solution, etc.). It is also possible for the field *value* to be empty (*value* = *none*, for example the global domain that does not contain any interface). The field called *node* can be an empty list, one child or a list of children. The *type* is a string that suggests the module or class to which the described node belongs. By convention and for easier recognition, it ends with *_t* (for example: 'Subdomain_t', 'Grid_t', 'Boundary_t', etc.).

Code organisation

The entire Navier–Stokes code is structured in 12 modules as shown in fig.3.1. We briefly describe each one of them.

Tools. This module is a collection of Python functions, tools that help building the initial tree structure. Since there is no standard way of generating sub-domains from a global domain, we choose to use the common approach consisting in first partitioning a domain in non-overlapping sub-domains (of equal size if there are no other indications), and then use a procedure to add overlap and/or ghost cells. The user is asked to define a list of parameters and a list of choices. He sets the global physical domain by its origins and length, or, for different reasons (complex geometries, avoid solid bodies, ask for different discretisations) sets main regions by their origins and lengths. Furthermore, each region can be split into any number of sub-domains. Once the sub-domains are created neighbours and boundaries are defined.

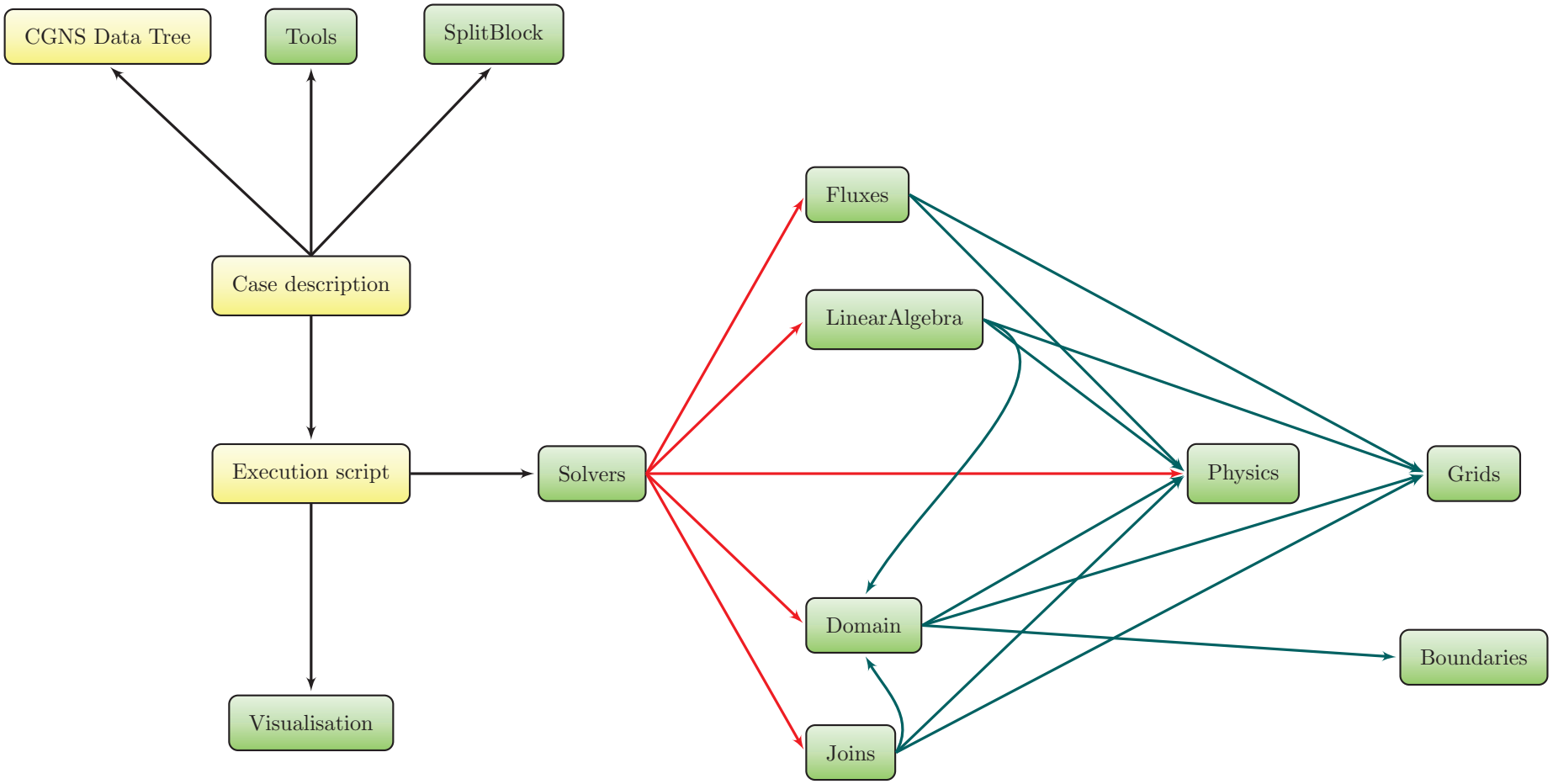


Figure 3.1: **Code organisation. Communication between structure modules (green) and Python files (yellow). Although each module is constructed so as to work independently, the complete Navier–Stokes platform is a joint work. Once the initial data tree is built, the main module is Solvers, containing all the solvers and in charge of all communications between computing modules, directly (Fluxes, LinearAlgebra, Physics, Domain, Joins) and indirectly (Grids, Boundaries). The detailed description of each module is given on pages 75 to 76.**

SplitBlock is the only imported module of this code. It has been build inside Onera in the 90's and used as a graph-partitioning for structured meshes into well space balanced sub-meshes. It takes into account the number of available processors and assigns one or several sub-domains to each processor. It can treat any type of boundary condition and decompose different levels of multi-meshes. The user can choose which domain to split or not. This module is used especially when matching meshes are used, for the scalability study and mostly in the case of the 2D isentropic vortex. It can be seen and used as a complementary module to the module Tools.

Visualisation. The solution is usually analysed under its primitive form. We use Paraview, an open-source, multi-platform data analysis and visualisation application for a qualitative and quantitative study of our solutions. In order to be readable by Paraview, the solution must be saved under an adequate form, as a vtkStructuredGrid file. Between the large amount of functionalities that are proposed by Paraview, we mostly use visualisation of reconstructed global domain, sub-domains, extract regions of sub-domains. One can choose between the visualisation of the density, the pressure, the acoustic pressure, the internal energy, the total energy, the velocity (u, v or/and w) and the vorticity.

Solvers is the central module of this code and it contains the collection of all explicit and implicit solvers described in the previous sections. It contains only cython files with classes that describe the second order Runge Kutta and second order BDF parallel solvers for each of the domain decomposition method presented on chapter II.

Fluxes. Inside this module one can find every definition and computation concerning the convective and the viscous fluxes, first order (section 1.2.3: van Leer, Steger Warming, LLF, HLLC, AUSM$^+$-up) and second order MUSCL numerical scheme to compute the convective fluxes and a second order numerical scheme to compute the viscous flux (see section 1.2.4).

LinearAlgebra. This module gathers three linear solvers: FGMRES (section 1.3.2), Gauss-Seidel (section 1.3.2), LDU-relaxation (section 1.3.2). The computation of the LDU-relaxation method is split into the computation of the spectral radius, different stages of relaxation: L-relax, D-relax, U-relax, error verification for both Euler and Navier Stokes equations on coincident and non-coincident meshes and different types of boundary conditions.

Physics. This module is built as a collection of primitive operations, constant definitions and memory allocations that are vital for the entire code. It contains the C/Cython declarations and C memory allocations of a general conservative field $U = (\rho, \rho u, \rho v, \rho w, \rho E)$; the constant coefficients defined by the state law (c_p, c_v, γ, Pr, K_T, Re, other coefficients can be added), basic operations with conservatives fields issued of CBLAS operations (copy, deepcopy, sum, extraction, multiplication or division of two conservative vectors,etc.). All constant coefficients have the default values of a perfect gas, but their values can be set before calling the solver. Conversions from primitive variables to conservative variables and from conservative variables to primitive ones are also possible inside the Physics module.

Domains gathers all information and operations concerning one sub-domain: memory allocations for flow variables, stock and update of the local solutions, computation of local time step from a CFL condition, links to local grid (origins, steps, etc.), boundaries and interface communications. The use of this module structure can be seen as a way of working and modifying global variable.

Joins. To be able to exchange data on the interface, each sub-domain (one sub-domain per process) must have a description of its interface. Joins is one of the largest module assembling all interface descriptions: neighbor sub-domain identification numbers, list of nodes to communicate and to be communicated, etc. For non-coincident meshes time and space quadratic interpolations are also defined in this module. The information of each transmission condition (Dirichlet or Robin type) is contained by both the source and the target sub-domain through their unique identification number.

Grids. Two kinds of Cartesian grid are defined (regular and irregular grid) and their characteristics: grid dimension, grid size, grid shape, origins, cell coordinates, volumes.

Boundaries. Beside the previously described inflow boundary, outflow, non-reflecting and reflecting boundary conditions are implemented (see section 1.2.2).

3.2 Parallelisation techniques

We begin this section by introducing two important notions often confused: process and processor. A processor is a hardware containing a central processing unit (CPU) capable of executing a program. A process is an instance of an application, a software concept, consisting in one or more threads, an executable, a private memory (PM) and sometimes a shared memory (SHM) (if needed by the system or by the developer). A tread or an active object is the smallest sequence of programmed instructions that can be managed independently by a scheduler. We can say about a program that it is executed in parallel if, at any time, more than one process, instances of this program, are concurrently active. In the next section, if not otherwise specified, we will consider that one process contains one thread.

Let us take a very popular example of one industrial application which illustrates the size and duration of one simulation with one process. To forecast the weather on the whole global atmosphere divided into cells of size 1 mile $\times$ 1 mile $\times$ 1 mile to a height of 10 miles (say 5×10^8 cells) over one day using 10-minute interval and supposing each calculation requires 200 flops, a computer operating at 1 GFlops would take over 8 days. The scale of this example is considered coarse, and can only provide an inaccurate picture of reality, and is too slow. A more advanced computer would diminish this computational cost. A modern computer can operate from 2 to 2600 GFlops (1 GFlops = 10^9flops), based on hardware configuration. Moreover, the number of floating operations per seconds is not the only factor to determine the speed of one simulation. One should consider the response time, number of pipelines (set of data processing elements connected in series), pipeline length, cache size, cache latency, number of registers and more. The fastest single computer record was detained in 2013 by China's Tianhe-2 with a record of 33.86 Petaflops (10^{24} floating point operations per seconds), but is of course unaffordable for most research institutes.

The available solution to complete such simulations, driven by real-time computing, is parallel computing. The efficiency of parallelism depends on hardware, algorithms and software. About hardware, the use of workstation networks to communicate between computers is proved to be more interesting than the use of one advanced single-processor computer that is more rare and expensive due to more complex architecture (NUMA architecture). If several computers are connected to each other via local area networks (LAN), a network referred to as a computer cluster is created. Computers within a cluster are referred to as nodes and the number of processes inside one node is the same as the number of cores or CPUs. We differentiate two kind of architectures: SIMD (single instruction, multiple data) and MIMD (multiple instruction,

multiple data). The SIMD device has a single unit devoted to control, but is typically performing the same instruction in parallel on different data. The MIMD device contains several decoder instructions which can either share the same RAM (shared memory MIMD) or access a collection of computing units interconnected by a network (distributed memory MIMD). On fig.3.2 one can see a typical memory model for n processes sharing the same global memory, they can point to different memory parts (different addresses of one vector), but also on exactly the same address.

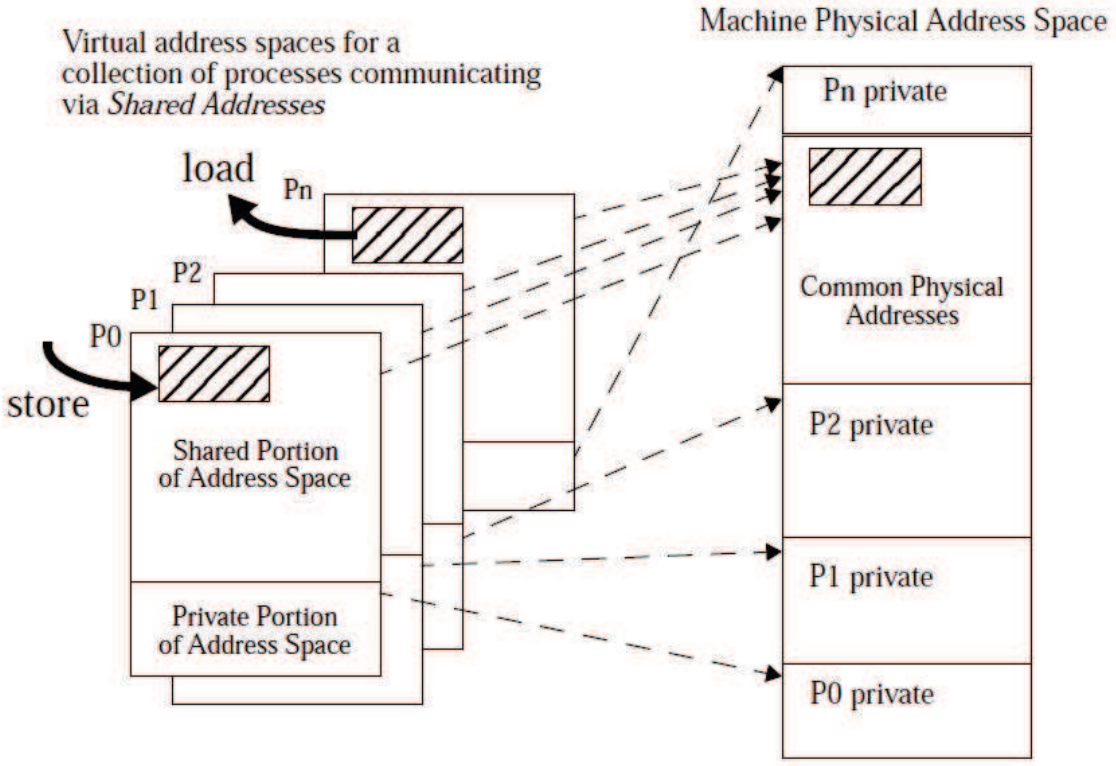


Figure 3.2: Memory model for shared memory parallel programs. P_n processes containing shared and private variables. Image source [26].

Our study first focuses on MIMD type architecture on which we differentiate two kinds of parallel computing: message passing parallelisation and loop parallelisation. These concepts are complementary to the programming languages and methods.

3.2.1 Parallel computing inside loops (OpenMP)

Inside loops, there are two possible automatic parallelisations using cyclic multi-threading (two threads work concurrently on two consecutive data) or pipeline multi-threading (two threads work concurrently on two blocks of consecutive data). OpenMP (Open Multi-Processing, [83]) is a parallel programming model for shared memory, it consists in a set of directive-based approaches. A program begins with a single thread of control called master thread. The calling of an OpenMP instance creates new threads of execution that can individually invoke program sub-parts without interfering with the other threads. The compiler is still the one responsible for producing the final executable code and must be an OpenMP-compliant compiler (Intel, Sun, gcc 4.0, etc.). When several threads share the same address, conflict and synchronisation issues may appear and must be managed by the developer.

The concept may not be obvious, but using OpenMP attributes to parallelise loops is quite simple. Let us consider the following example of a C program that computes a numerical Euler flux using the AUSM$^+$−up scheme. In C language the command *# pragma omp parallel for* is

telling the compiler to parallelise the first loop that occurs. If we do not specify the number of threads we want to create (*num_threads*(*th_number*)), the compiler will try to create and use all available threads.

```
. . .
# pragma omp parallel for private(i) schedule(static,10)
  for ( j = 0; j < ncj; j++) {
    double rl,ul, vl, wl, aul, pl, hl, hlt, asl2, asl, ahl;
    double aint, Ml, Mr, Mb2, t, M02, M0, fa;
    double Mp, Mm, Mint, mint, p5p, p5m, pint;
    double qn, invRho, rhoint, alpha;
    CfdField ptLi, ptRi;
    for (i = 0; i < ninti; i++) {
      /*---------- Left state ---------*/
      rl     = (*ptLi.pt_rho);
      invRho = 1./rl;
      /* Direction component of the left flow speed */
      ul = invRho*(*ptLi.pt_rhoU);
      aul= fabs(ul);
      vl = invRho*(*ptLi.pt_rhoV);
      wl = invRho*(*ptLi.pt_rhoW);
      /* Compute left pression number */
      qn= 0.5*(ul*ul+vl*vl+wl*wl);
      pl = eos->gm1*((*ptLi.pt_rhoE)-rl*qn);
      /* Left enthalpy */
      hl = ((*ptLi.pt_rhoE)+pl)*invRho;
      hlt= hl + qn;
      /* Left sound speed */
      asl2= self->tgm1ogp1*hlt;
      asl = sqrt(asl2);
      /* Numeric left sound speed */
      ahl = asl2/Max(asl,aul);
      /*-------- Right state -----------*/
      ...
    }
  }
 ...
```

The command *private(i)* asks the compiler to make uninitialized copies of i with the same type and name as the original variable. Otherwise, all variables defined after the line *# pragma omp parallel for* are new local variables to each thread. One can also use a common variable to each thread using the command *shared*(*variable*). *schedule*(*static*, m) is the default command used to sequentially distribute equal-sized loop items (loop_count/ number _ of _threads), execute m iterations (or less for the last series), then ask for another set of m. Other options are also available: *dynamic*, *guided*, *auto* and *runtime*.

We use OpenMP only for loop parallelisation, but its concept goes beyond that; other tasks (task farming technique) or sections can also be done in parallel. In the case of loops parallelisation via OpenMP there is no need of a synchronisation command, an implicit barrier is done at the end of the loop. Beside helping parallelising small parts of applications, the use of OpenMP is very attractive because only small modifications of the sequential code are necessary, the expression of parallelism is clear, the code size grows only modestly. An important remark is that the amount of operations inside one thread must be much more important than the amount of read data, otherwise the overhead of the data access (saturation of the memory bus) may be greater than the cost of doing the work on one thread.

3.2.2 Parallel computing via message passing (MPI)

The MPI (Message Passing Interface, [75]) is a public library based on the Message Passing concept. It provides an interface, independent of computer architecture, allowing communications between tasks for distributed memory processors, shared memory processors, networks and workstations. The scientists implement and compile their codes as usual and link them with the MPI library. The processors communicate among each other by sending and receiving the so-called transmission conditions.

In our computation, each sub-domain is assigned to one and only one node (even if different approaches are possible). The assignment is done inside the module Tools at the moment of the sub-domain construction by adding a tree node containing a unique number equal to the rank of one processor. Different levels of one algorithm (depending on the numerical scheme) are computed locally by one processor. But, to coordinate the data exchanges and verify the convergence behaviour, a global administration component has to be defined. The developer must keep track of memory usage. Our strategy is to use as many non-blocking MPI commands as possible to overlap message passing and computation. Even though we do our best to well-balance the sub-domains and locally discretise in time, some processors may still finish one job before another. When interface conditions must be transmitted from one processor to another, the first one sends its data as soon as it is available and the second one receives it as soon as it is available. These commands allow an overlap of tasks within the moment of a send and of a receive of information, which is the best possible scenario.

MPI is supported by all programming languages (Fortran, C, C++, Python) and is known for portability and performance when it is well used. For Python, several MPI libraries exist, such as pyMPI and mpi4py. The mpi4py library, more efficient than pyMPI, was initialised in 2006 by L.Dalcin. It is written in cython and enables distributed memory paralleling. Let us take the following example of parallel cython code (identical to Python). First, we need to initialize the MPI environment using $MPI.COMM\_WORLD$. This sets all available processes at start-up time and their communication context.

```
...
import mpi4py.MPI as MPI
/*-------- initialisation  --------*/
comm    = MPI.COMM_WORLD
// get internal process number
rank    = comm.rank
// get number of available processes
nb_proc = comm.size
...
if rank == rankTarget:
  buff = [numpy.empty(dim)]
  /*  set non-blocking receive    */
  rho  = comm.Irecv([buff[0],MPI.DOUBLE],source=rankSource,tag=2*
      self.domSrc+8192*self.domTgt)
  /* compute local time step  */
  dim      = ssdom.wConservative[0].dimension
  dt,mindt = ssdom.computeLocalTimeStep(self.eos,dir=0,cflCoef,
      dimCellPh,U)
  dTMinLoc = min(mindt,dTMinLoc)
  if dim > 1 :
    dt,mindt = ssbdom.computeLocalTimeStep(self.eos,dir=1,cflCoef,
        dimCellPh,V)
    dTMinLoc = min(mindt,dTMinLoc)
    if dim > 2 :
```

```
            dt,mindt = ssdom.computeLocalTimeStep(self.eos, dir=1,cflCoef
                ,dimCellPh,W)
            dTMinLoc = min(mindt,dTMinLoc)

        /* wait for rho to be received */
        MPI.Wait(rho)
        solveBDF(initSol, rho, rhoU, rhoV, rhoW, rhoE)
        [rho, rhoU, rhoV, rhoW, rhoE] = updateSolution (rho, rhoU, rhoV,
            rhoW, rhoE)
        ...
      if rank !=rankSource:
        ...
        /* blocking synchronous send  */
        send_rho = comm.Ssend([numpy.array(rho) ,MPI.DOUBLE],dest=
            rankTarget,tag=2*self.domSrc+8192*self.domTgt)
        ...
    ...
```

Suppose that we solve the Navier–Stokes system of equations on a bounded domain Ω split into two overlapping sub-domains Ω_1 and Ω_2, and that Ω_1 (source) sends an array to Ω_2 (target). We proceed as follows:

- the source sends a blocking synchronous mode array (*comm.Ssend*). It means that Ω_1 begins sending a message (rho) even before checking if there is a matching process waiting for an information (through tag matching). However, the send cannot complete until its matching receive is found, and the receive operation has started to receive the message. If both sending and receiving operations are blocking synchronous modes, the communication is also synchronous;
- the target receives the message in a non-blocking way (*comm.Irecv*). The system can start writing data into the receive buffer. Meanwhile the target can continue working (computing the local time step in the example) as long as it does not access this buffer. When the receive data is needed, we make sure that the receive is finished using the MPI_Wait command, which also deallocates the receive buffer.

This choice of communication routines seems appropriate to our strategy as it allows overlap between tasks. In the case of distributed memory parallelisation, the amount of operations inside one process must be much more important than the communication between processes, otherwise the overhead of the communication may be greater than the cost of doing the work on one thread.

MPI is considered optimal for process level parallelism (private memory) and OpenMP for loop-level parallelism (shared memory). Combining them, we construct a hybrid program on multiple levels of parallelism and reduce computational cost. The presented approaches are suitable to hybrid approaches if one sub-domain is assigned to one processor (distributed memory computing via MPI) and local loops are computed in parallel on shared memory (OpenMP). The portability and the robustness of the parallel code should be comparable with those of the sequential code.

3.2.3 Graphic Processor Unit (GPU)

Thanks to the game industry, recent graphical cards raise the computing capabilities, with higher performance than classical CPUs (fig.3.3).

When Pixar introduced shaders in Renderman software in 1988 and in OpenGL library version 1.4 in 2001, scientists began to use graphical cards to compute numerical simulations. Programming with shaders is not well adapted to numerical simulations. NVIDIA with CUDA

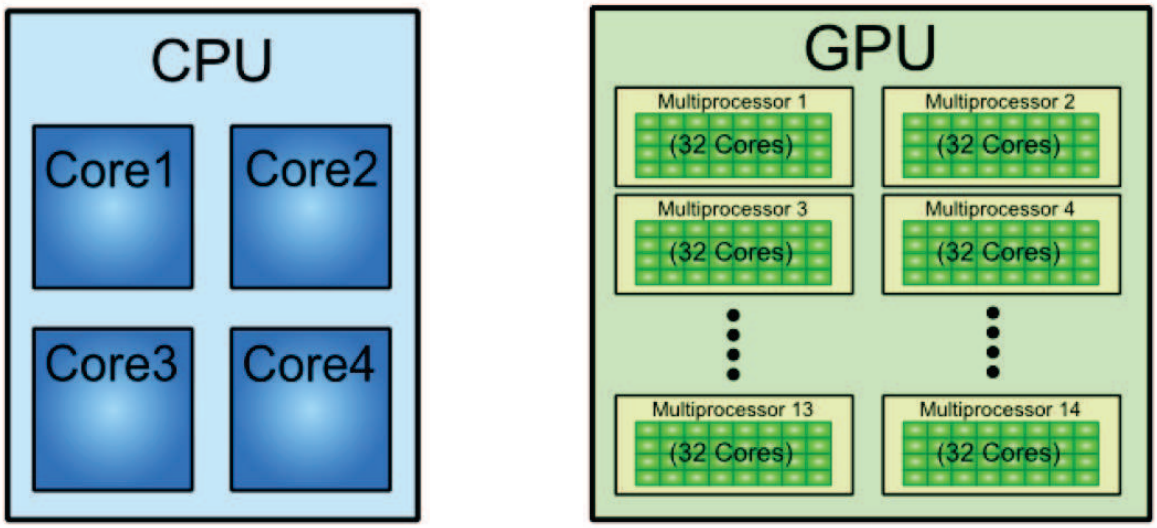


Figure 3.3: The number of cores inside a CPU versus the number of cores inside a GPU.

([25]) and AMD with Radeon SDK, the leaders among GPU constructors, provided more flexible tools for their GPUs. It was the Khronos group that later tried to unify the GPU programming with OpenCL standard. Recently, PGI, NVIDIA and Cray proposed the OpenACC standard and the OpenMP consortium proposed a new version of its standard, including some directive to compute on GPU devices. None of them are available in current compilers. Nevertheless, a first beta version of some popular compilers, such as CGG *(https://gcc.gnu.org/wiki/openmp)*, are implementing OpenACC and OpenMP 4.0 features to use GPU devices.

CUDA and OpenCL are parallel computing platforms, application programming interfaces (APIs) usable with C and C++ language providing direct access to the memory of GPUs. They can be seen as a extension of the C and C++ programming languages. Both of them dispose of a Python interface: pyCUDA and pyCL. OpenCL was designed to work on all GPU cards: AMD, NVIDIA, Intel, S3 Graphics, Matrox, Texas Instrument, etc. Although, the OpenCL code is portable, for best performance, it must adapt to all architecture types, and so the developer is faced with many options and ways to code memory allocations, transfers, etc. Moreover, OpenCL is properly supported only by AMD; the NVIDIA support is concentrated only on CUDA. We will especially refer to GPGPU (Global Purpose Graphic Processor Unit) of NVIDIA and discuss programming with CUDA (Compute Unified Device Architecture), but note that CUDA programming is similar to OpenCL and it can be ported over to OpenCL.

The NVIDIA GPUs architecture is referred to as SIMT (single instruction, multiple threads, [25, 77]) and is closely related to SIMD and shared memory MIMD (OpenMP). In SIMD architecture a single CPU devoted to control is capable to process in parallel elements of vectors. The SIMT architecture is more flexible than SIMD as it can additionally run:

- a single instruction on multiple register sets: each thread contains multiple registers that process different parts of an instruction;
- a single instruction on multiple addresses: parallel random access with memory (only parallel random access with registers is possible on SIMD);
- a single instruction on multiple flow paths: conditional jumps for different threads.

In shared memory MIMD architecture, instructions of several threads can run in parallel. SIMT is less flexible than shared memory MIMD. Inside a SIMT architecture local synchronisation can be done inside a block; a global synchronisation must be manually implemented.

When a new architecture appears, a developer must re-evaluate, rethink the problem and be able to appreciate its adaptivity to this architecture. The aim of this section is to render the

GPU architecture comprehensible to the reader so he can appreciate its utility. Through the entire section, we highlight the advantages and the limits of this powerful parallel tool.

Hierarchy inside a GPGPU of NVIDIA

Different from a CPU architecture, a GPGPU contains three memories. The global memory is used to communicate with the host (key word for CPU). The texture and the constant memories contain constants initialised by the host. Only the texture and the constant memories dispose of a cached memory. The host can access in read and write (R/W) mode any of the three memories. The three memories are visible to all threads. The global memory can be accessed by all threads in R/W mode, the texture and constant memories can be accessed by threads only in read mode. The texture memory can be accessed in R/W mode through surface objects thanks to the API.

The memory of a GPGPU, on fig.3.4, is structured in grids of blocks of threads. A grid is a set of blocks that can be executed in parallel. A block of threads can have different size and contains threads that cooperate with each other through shared memory and local synchronisations. Each thread has a local memory, can access shared memory inside a block and has access to the global memory. A very important remark is that, unlike inside a CPU, where all threads can cooperate, the threads between two different GPU blocks do not cooperate as they do not share the same shared memory. Moreover, a block is split in equal sized groups of threads called warps (usually of 32 threads). The splitting is done in a consecutive way, thread number 0 is inside the first warp. The separation into warps is not directly visible to the developer, yet it will have a major impact on the computation as access to the global memory is done by warps.

Let us summarise on the hierarchy inside a GPU : grids $\rightarrow$blocks ($\rightarrow$ warps)$\rightarrow$threads. Each grid and block can have one, two or three dimensions. Each block and thread have a unique id: $blockIdx.\{x, y, z\}, threadIdx.\{x, y, z\}$.

The multiple levels of threads, memory, and synchronization are suited to the so called fine-grained data parallelism. Inside one block, all threads work in parallel. Blocks can either work in parallel or sequentially, depending on the GPU resources. Parallel sections of one algorithm are executed on the device (key word for GPU) as kernels which run in parallel on many threads. By default kernels are sequentially launched. Inside one block, threads can be manually synchronized using the command $\_\_syncthreads()$ that acts like a barrier. If threads access memory in an uncoalesced manner the computation slows down. The access to the shared memory must be coordinated, each warp should access continuous regions of the memory and avoid bank conflicts. The cache memory is structured in memory banks. A memory bank is a unit that is addressed consecutively in the total set of memory banks. When more than one thread of the same warp access the same memory bank the computation is also slowed down. Also, if two threads write at the same address a conflict occurs and the accesses will be serialised.

Definitely the most attractive property of the GPU is that it contains hundreds of cores and millions of independent threads that can work in parallel (against a few on a CPU). For example, one block can contain up to 1024 threads on a NVIDIA Tesla architecture. Similar to OpenMP, a CUDA program runs on any number of processors. The execution of kernels on the GPU and transfers from or to the GPU can be done in the same time. Also, the GPU is cheap and available. A GPU can access data on the CPU via PCI-Express, but the access is much more expensive than global memory accesses. Large packages of data, but less than 6 GB should be copied to the GPU. The GPU must have enough work to do to become efficient and needs explicit synchronization for data coherency.

To be able to efficiently program on GPU, one should avoid conditionals and loops in kernels. Duplicating computations can be faster than to compute on one thread and broadcast the results to the others. Communications are costly!

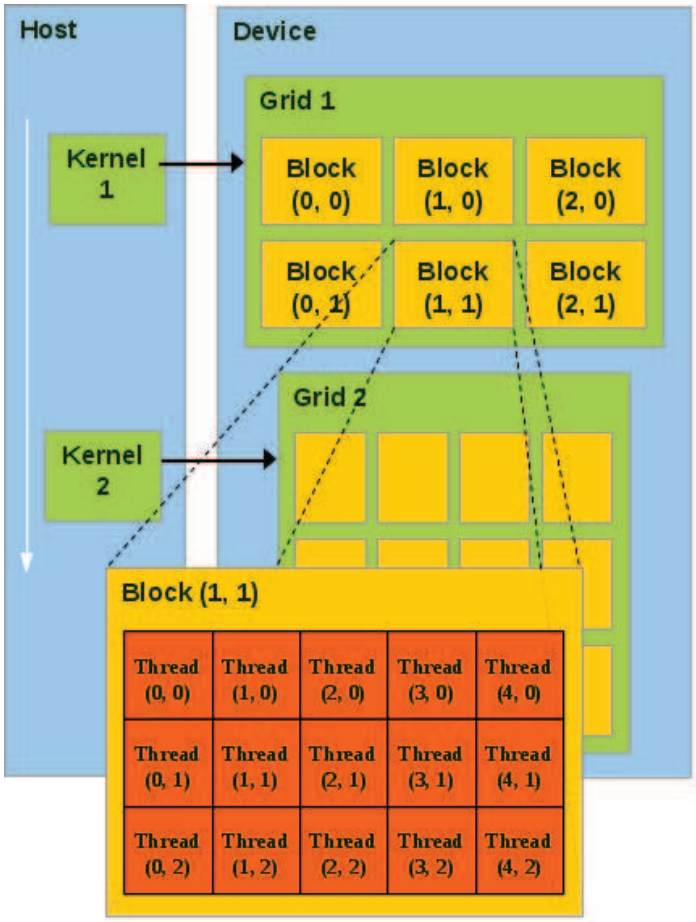


Figure 3.4: Memory model for shared-memory GPU. The host denotes the CPU, parallel kernels are sequentially launched on device (GPU). The device contains several grids, each grid is split in blocks of threads. A thread is the smallest active object able to execute an instruction. Image source GPU summer courses [26].

High performance means exploiting all levels of parallelism and leads to the idea of combining the three discussed architectures, thus the use of OpenMP, MPI and GPU (see one possible configuration of combining all three architectures on fig.3.5). The communication between devices become the biggest issue in achieving performance. Multiple GPUs are possible, the same host can access two or more GPUs, but the selection between GPUs must be manually controlled. CUDA is message passing between CPU and GPU. CUDA and MPI are complementary, there is no major difference between a classical MPI on multiple CPUs and MPI on GPUs. OpenMP+MPI+GPU adapts well to codes that compute large, disjoint problems simultaneously.

The time spent to learn and implement on GPU is much higher than its equivalent on CPU. Many researchers would consider that GPU programming is complicated and will not dare to learn it. An alternative exists, the use of available GPU libraries. Based on the idea: do not *reinvent the wheel*, developers propose freely the use of their optimized codes on GPU to solve partial differential equations, systems of linear equations (dense or sparse), preconditioning of large systems, computation of eigenvalues/singular values of matrices, partitioning large graphs, non-linear systems, etc. A list of libraries and their main applications can be found in Appendix B.

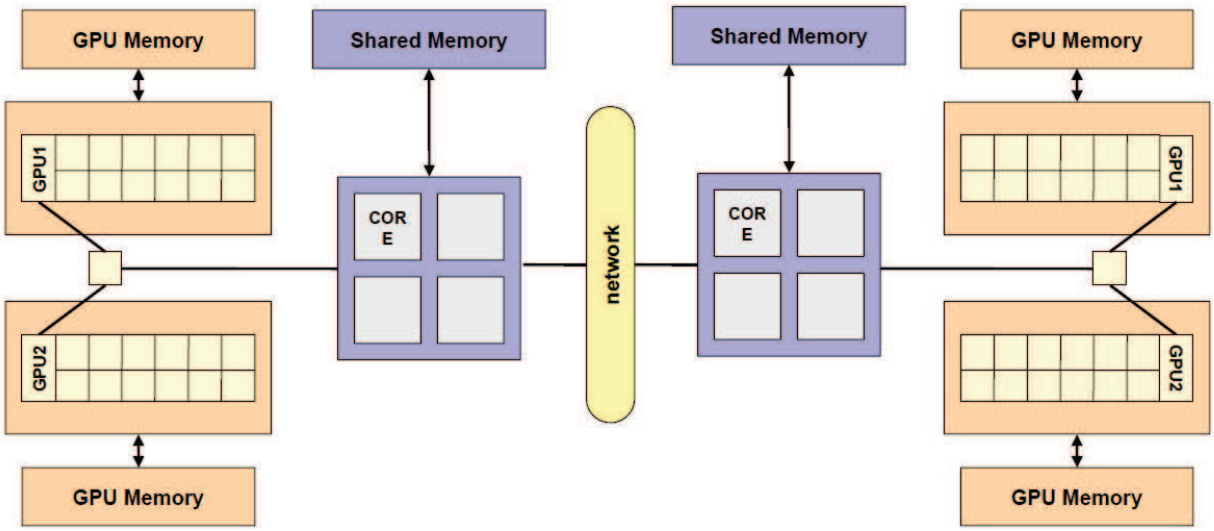


Figure 3.5: Possible architecture combining all levels of parallelism: OpenMP + MPI + GPU. The communication is done by the networking inside clusters (MPI), each node parallelise tasks between cores (OpenMP) and each core can parallelise kernels inside a GPU. Image source GPU courses by CAPS.

Discussions on additional difficulties of GPU programming

Due to coalescence and bank conflict problems in the shared memory, an optimal implementation of stencils is a true challenge on the GPU device. The more compact the stencil, the easier it is to avoid uncoalesced memory access. Moreover, computing structured meshes is easier than unstructured meshes. Because of these reasons the Navier–Stokes system of equations is a challenge (see the PhD of M.Lefebvre [61] for applications to the Navier–Stokes equations). We propose here a discussion on the implemented CUDA code to solve the Euler equations using a mono-domain second order explicit Runge-Kutta method.

The last module added to the new code is the cudaGPU module. As mentioned in the beginning of this section, the entire problem must be rethought. Thus, each presented module was rebuilt and adapted to GPU with CUDA. The first step in developing an Euler solver on GPU was to create a class of array methods that allocates GPU memory for an array of different dimensions and lengths. We used *cudaMalloc* command for 1D arrays and *cudaMallocPitch* for 2D arrays. *cudaMallocPitch* is best for 2D memory allocations as it makes sure that the starting address of each row in the 2D array (row-major) is a multiple of 2^n (n depending on the compute capability) and facilitates the shared memory access by warps. The data transfer between CPU and GPU is made via *cudaMemcpy2D* or *cudaMemcpy* calls which are able to transfer data between GPU devices if *cudaMemcpyDeviceToDevice* option is active, from CPU to GPU if *cudaMemcpyHostToDevice* option is active and from GPU to CPU if *cudaMemcpyDeviceToHost* option is active. Another tool file is constructed to define the number of available devices, the devices names, their architecture, generation, available cores and capability.

Inside a CUDA file (present in each module), a kernel is recognized by the key word *__global__* that precede its definition. Functions executed and accessed locally on the device are preceded by the key word *__device__*. For a CPU to execute a kernel, it must first define its size (grid size and block size) that can not exceed the maximum possible size on a GPU (see example of kernel call below where *dim*3 denotes a structure of three integers).

```
__global__ myKernel(arguments)
{
. . .
}
void myCPU(arg)
{
  dim3 threads(sizeGrid, sizeGrid, 1);
  dim3 grids((nci+sizeGrid-1)/sizeGrid,(ncj+sizeGrid-1)/sizeGrid,1);
  myKernel<float><<<grid,threads>>>(arguments);
  ...
}
```

We propose to briefly discuss on two steps inside the computation: the time step computation and the solver computation.

1. Time step computation

In general, the time step inside a domain Ω (see 1.3.1 for details over the stability criteria) is given by the following formula: $\Delta t = \min_i(\Delta t_{C_i}, \Delta t_{D_i}),\ i \in \Omega$. For the Euler system $\Delta t = \min_i(\Delta t_{C_i})$. A minimum over the entire velocity field must be computed. As seen below, for one mesh cell, the minimum function is equivalent to the one on the CPU:

```
// ------------------------------------------------------
template<typename K> __device__ K
gpu_compute_dt(int dir,K gamma,K mu, K Pr, K cfl, K dx,  K rho, K rhoU,
     K rhoV, K rhoW, K rhoE)
{
  K rho_i, u[3], p, a, dtC,dtD;
  rho_i = 1./rho;
  u[0]  = rho_i*rhoU;
  u[1]  = rho_i*rhoV;
  u[2]  = rho_i*rhoW;
  p     = (gamma-1.)*(rhoE-0.5*rho*(u[0]*u[0]+u[1]*u[1]+u[2]*u[2]));
  a     = sqrt(gamma*p*rho_i);

  dtC   = cfl*dx/(fabs(u[dir])+a) ;
  dtD   = 0.5*cfl*dx*dx*rho*Pr/(mu*gamma);

  return gpu_min(dtC,dtD);
}
```

The execution of the minimum is visible only by the device and done by each active thread. To find the global minimum, we use a parallel reduction, close to the one presented by D. Negrut [76] in his GPU course. We present here the steps needed for an optimal, fully parallel reduction. The entire code is presented in annexe C.

1. Implement a global GPU function (*gpu_reduce_min*) that reduces, for each two blocks, the minimum of the known vector (computed with *gpu_compute_dt*). This first step reduces the number of blocks to its half. By repeating this step we reduce the computation to one block of minimum values.
2. Compute *gpu_warpReduce*, a function that unrolls the last warp inside the last block. As reduction proceeds, the number of active threads decreases (for $tid < 32$ we only have one warp left). Unrolling the last warp avoids using the thread synchronisation

(_syncthreads()) and saves useless work in all warps. This step is done also to avoid uncoalesced access to the cache memory. The number of threads in a block is given at the compile time (and they are power of 2 block size and limited at 1024 threads on Fermi) and allows us to completely unroll the reduction. Yet, the developer should consider a generalised formulation for different number of blocks.

When we compute a minimum over a 2-dimension vector, we choose a privileged direction and similarly proceed. Note that parallel reductions are also used on CPU computation to diminish costs, but on GPU its utilisation is vital.

Second order Runge Kutta method

We do not recall the numerical explicit Runge Kutta method, one can see 1.3.1 for details. We discuss only the need to exploit the GPU shared memory. Let us recall the middle step in the x-direction of the second order Runge Kutta method applied to the Euler Equations and for one cell Ω_i.

$$U_i^{n+\frac{1}{2}} = U_i^n + \frac{\Delta t}{2}(F^n_{Euler_{i+\frac{1}{2}}} - F^n_{Euler_{i-\frac{1}{2}}}).$$

First, we compute the convective one order fluxes using an AUSM$^+$−up scheme. To compute $F^n_{Euler_{i+\frac{1}{2}}}$ we need to access the conservatives field in i and $i+1$. Then for the right part of the Flux over a block of size $blockSize * blockSize$ we need to access a block of size $(blockSize+1) * blockSize$. So, one additional column has to be accessed on the global GPU memory (slower than the access on the shared memory). Moreover conflicts of access and writing may appear. We use then the shared memory to store the necessary values. Secondly, second order convective flows are computed with a MUSCL scheme. This one requires the values of first order fluxes for a 3 stencil size ($i-1, i$ and $i+1$). A 2D array of size $(blockSize+2) * blockSize$ must be stored in the shared memory. In a similar way, to advance in time we need to use the shared memory for a block of size $(blockSize+1) * blockSize$. Dirichlet boundary conditions are most adapted to GPU programming since there is no need to access any neighbour cell.

For unstructured meshes, more interaction between blocks, memory copies and used of the shared memory is required.

3.3 Parallelism efficiency evaluation

We can see parallel computing only from a technical point of view, but the efficient way is to develop a decomposition strategy on the initial mathematical problem. When parallel computation is done one sub-domain is affected to one specific processor. This processor handles both data (geometry, coefficients) and the computations. There are different ways of measuring the performance of a parallel computation. From a strictly technical point of view we distinguish three metrics: the latency or the time to execute one operation (measured in operations per seconds), the bandwidth or the rate at which the operations are performed (measured in seconds per operation) and the cost, equal to the product of the latency with the number of performed operations. For a non-parallel process, instructions are sequentially computed and the entire data is local, then these metrics give enough information. But, for parallel computing, processes and processors need to access non-local data, they usually communicate among each other by sending and receiving the so-called transmission conditions. Even though techniques that synchronise the transfers exist, processors may need to wait for an information before being able to proceed to another operation. Then, the global cost must contain the communication

cost:

$$\text{CommunicationCost= frequency}\times\text{(CommunicationTime-overlap)}.$$

The frequency is the number of communication operations per unit of work and it is usually specific to an architecture and can be found in the computer specifications. The communication time contains the time needed for the processor to initiate the transfer and the necessary time for the data to travel in the communication path. Between the time of the transfer initiation and the receive of information, one processor may keep performing other operations, this portion of time is called overlap. For more details on communication measuring we refer to [26].

We focus on a global way to measure the effort to solve a problem in parallel. Meaning, we consider that the total computational cost is the cost of reaching the convergence criteria in each sub-domain added to the synchronisation and the communication cost:

$$\begin{aligned}\text{cost} = \#\text{ iterations } \times \sum_i \text{ (cost of solving sub-domain i on processor i)} \\ + \text{ synchronisation cost } + \text{ communication cost .}\end{aligned}$$

We can now define a tool to verify the performance for both the architecture and the application developer called speedup. Assuming we use p processors, for a fixed problem the speedup is defined:

$$\text{speedup(p processors)} = \frac{\text{cost(1 processor)}}{\text{cost(p processors)}}.$$

For the speedup to reach the maximum value p, the synchronisation and communication costs must be very small and depend on the scaling efficiency of tasks. We introduce two basic ways of measuring the scalability or the scaling efficiency, the weak scalability and the strong scalability.

Weak Scaling

The size of each sub-domain is fixed and the number of processors involved in the parallelisation increases. This means that, when p sub-domains are computed in parallel the size of the global domain becomes p times the fixed size of a sub-domain, and therefore the space step diminishes. We give the weak scaling efficiency formula:

$$\frac{t_1}{t_p} \times 100\%,$$

where t_1 is defined as the amount of time (cost) to complete a work unit with one processing element, and t_p is the amount of time to complete p identical work units with p processing elements. Supposing that the physical size of the domain is $[0,1]\times[0,1]$, the size of the system to solve is $n \times n$ (see fig.3.6), t_1 is the amount of time to solve one system of size $n \times n$ with the space steps $\Delta x = \frac{1}{n}$ and $\Delta y = \frac{1}{n}$; and t_p is the amount of time needed to solve in parallel $p = p_x \times p_y$ systems of size $n \times n$, but with the space steps $\Delta x = \frac{1}{p_x n}$ and $\Delta y = \frac{1}{p_y n}$. We achieve full efficiency when t_1 equals t_p. It is the case where the cost of the exchange data between the sub-domains can be neglected and when the diminishing space steps does not influence the computation. It is the ideal case.

Strong Scaling

The second way to measure the scalability is the strong scaling. This time, the global problem size is fixed and we increase the number of processing elements. We recall that in this work only one sub-domain per processor is considered. When the number of processors increases

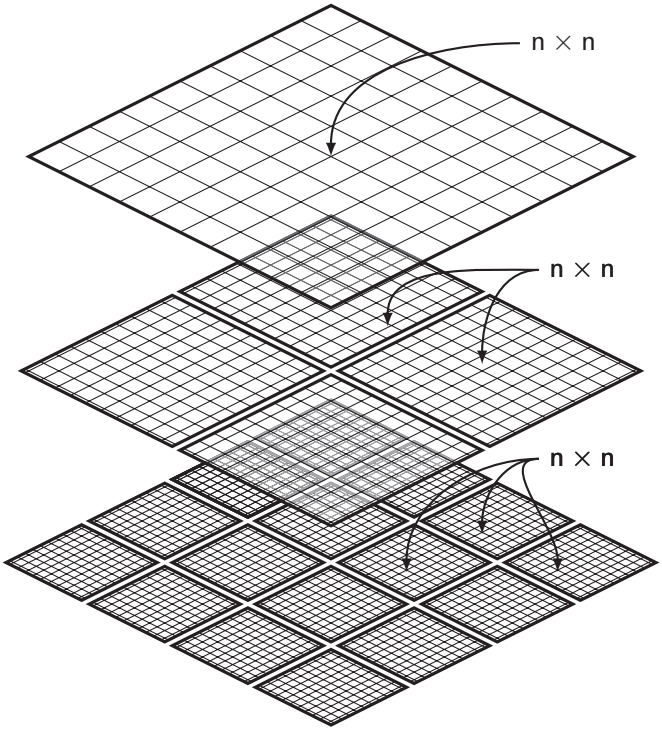


Figure 3.6: Weak scaling. The global computational domain Ω is of size $n \times n$. When on 4 (respectively 16) sub-domains are used to compute the same physical sub-domain, the mesh is refined in such a way to result in equal sized sub-domains of size $n \times n$.

the number of sub-domains increases and, this time, the size of a sub-domain diminishes. The formula of the strong scaling efficiency is:

$$\frac{t_1}{p \times t_p} \times 100\%.$$

Let $[0,1] \times [0,1]$ be the physical size of the computation domain discretised in $n \times n$ cells (see fig.3.7).

When $p = p_x \times p_y$ sub-domains are solved in parallel the size of one sub-domain equals $\frac{n}{p_x} \times \frac{n}{p_y}$ and the space steps are $\Delta x = \frac{1}{n}$ and $\Delta y = \frac{1}{n}$, the same space steps as when only one processor is in use. The ideal case is when t_1 equals $p \times t_p$ and it happens when the amount of time needed to solve one sub-domain is exactly $\frac{1}{p}$ the amount of time needed to solve the global system. The sub-domain solver should converge in exactly the same number of iterations as the global one and the number of data exchanges should be neglected.

The previous parallelisation efficiency evaluation can be directly applied to a distributed memory architecture via MPI parallelisation, but can also be adapted to the shared memory architecture. In our case, the loops are equal sized distributed, as in the strong scaling evaluation.

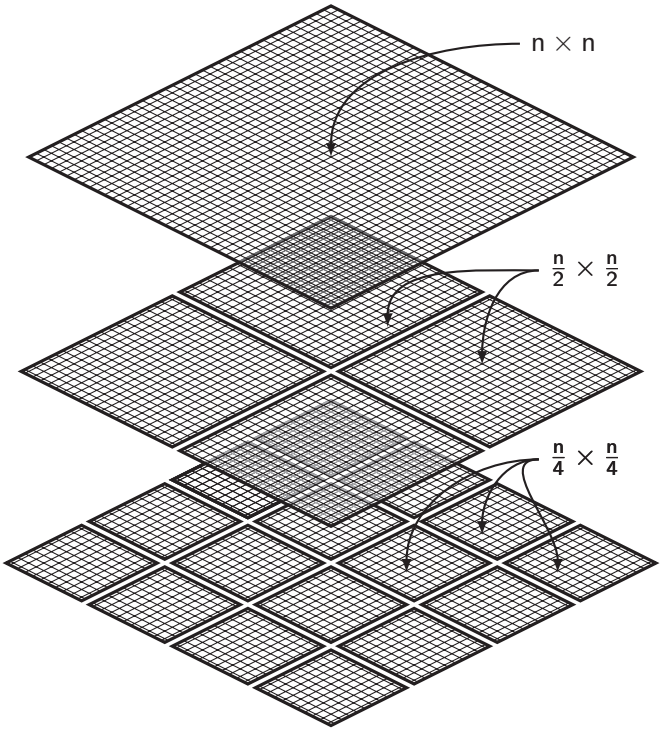


Figure 3.7: Strong scaling. The global computational domain Ω is of size $n \times n$. When on 4 (respectively 16) sub-domains are used to compute the same physical sub-domain, the mesh split in equal sized sub-domains.

Chapter 4

Numerical results and discussions

Contents

This chapter is dedicated to the validation of the presented methods. Firstly, validations and comparisons of different numerical schemes to compute the Euler fluxes are conducted. Several 1D and 2D configurations with shocks are chosen to verify the sensibility of the numerical schemes. The aim is to find, within the existing numerical schemes the ones suitable to our applications.

Secondly, we validate the classical domain decomposition methods applied to explicit and implicit 2D applications. Then, we focus on proving the robustness of the improved scheme on different number of sub-domains. At Onera, the PC cluster Madmax contains over 1100 cores within 75 nodes with a total of over 25To of storage and 2.5 To RAM and it is the cluster used to compute our simulations. The Madmax nodes can have different capacities and number of cores.

The first case is the isentropic vortex evolution based on Yee's paper [99] and it is very interesting for our applications since it provides an exact solution to the Euler system of equations. Moreover, when dealing with simulations of aircraft trailing vortices, blade–vortex interaction of helicopter rotors, rotor–stator interaction of turbo-shaft engines, aeroacoustic problems or weather forecasting, many computational fluid dynamics researchers are faced with the problem of numerical diffusion of vortices.

The second case is the sound generation in a 2D low Reynolds mixing layer, a very unsteady and sensitive case and intensively studied in the aeroacoustic field.

The last case is the vortex shedding around rectangles. It is the closest case to real life simulations. It has multiple applications [28, 9, 80, 92, 86] such as the aerodynamic drag reduction for air-planes, road vehicle, damage predictions for inclined air-foils, ocean pipe line or risers, off- shore platform supports, suspension bridges, steel towers or smoke stacks, etc.

4.1 Comparisons of numerical convective fluxes schemes

In order to compute the convective fluxes we have studied five different first order schemes: Godunov Scheme, van Leer Scheme, AUSM$^+$-up Scheme, HLLC Scheme, and LLF Scheme. Each scheme has intensively been studied in literature [89, 34, 98, 74, 95, 96, 97, 56, 69, 68, 67, 70, 93, 94] and has proved its efficiency for sharp non-linear solution structures such as shocks and discontinuities. To achieve second order accuracy the MUSCL strategy has been used. In order to validate these schemes, we start by performing one dimensional cases on shock tube problems presented by Toro in [93].

4.1.1 1D shock tube problem

In table 4.1 one can find the initial data for seven tests that we have chosen to study. For all tests the spatial computational domain is $[0, 1]$ and it is discretised with $N = 100$ cells. The fluid is a perfect gas and the diaphragm that initially separates two gazes with different properties is placed at x_0. The sudden removal of the diaphragm at time $t = 0$ creates a discontinuity between the two gazes. The resulting waves propagate in the tube and can be rarefaction waves, contact discontinuities or shock discontinuities.

These tests allow us to check the correct implementation of the schemes, they provide information about the robustness and the accuracy of approximate Riemann schemes. The CFL number is fixed to 0.7 and the boundaries conditions are of transmission type. For each test we can compute the exact solution by solving directly the associated Riemann problem. The resulting quantities of interest that we compare are the density ρ, the velocity u, the pressure p and the specific internal energy e.

The following results are obtained after solving the Euler equations with a first order Euler Explicit scheme. All figures show a comparison between the computed solution of the quantities

Table 4.1: Data of 1D test problems with exact solution

	Left data			Right data			Diaphram position	Stop Time
Test	ρ_L	u_L	p_L	ρ_R	u_R	p_R	x_0	timeOut
1	1.0	0.	1.0	0.125	0.0	0.1	0.3	0.25
2	1.0	0.75	1.0	0.125	0.0	0.1	0.3	0.2
3	1.0	-2.0	0.4	1.0	2.0	0.4	0.5	0.15
4	1.0	0.0	1000.0	1.0	0.0	0.01	0.5	0.012
5	1.0	0.0	0.01	1.0	0.0	100.0	0.4	0.035
6	5.99924	19.5975	460.894	5.99242	-6.19633	46.095	0.4	0.035
7	1.0	-19.59745	1000.	1.0	-19.59745	0.01	0.8	0.012

of interest at different output time units, presented on 4.1, and the exact solution, shown in red.

The first test is the so-called Sod problem ([89]) and it is a common test for the accuracy of Riemann based scheme. Its solution consists of three characteristics describing a left rarefaction, a contact and a right shock. The second test is a modified version of the Sod problem. The solution consists of a right shock wave, a right travelling contact wave and a left sonic rarefaction wave. The third test is the so-called *123 problem* ([31]) and it consists of two strong rarefactions and a trivial stationary contact discontinuity.

Test 4 consists of a left rarefaction, a contact and a right shock wave, it is the left half of the Woodward and Colella (see [98] for details) problem, called the Blast Wave Problem. The blast wave problem is physically composed of a tube containing three gazes with different properties that are initially separated by two diaphragms. Test 5 represents the right half of the blast wave problem of Woodward and Colella and is made of a left shock, a contact discontinuity and a right rarefaction. Test 6 is a combination of the resulting shocks of test 4 and 5. It solution represents the collision of these two strong shocks and consists of a left facing shock, a right travelling contact discontinuity and a right travelling shock wave.

Test 7 consists of a left rarefaction wave, a right-travelling shock wave and a stationary contact discontinuity. It is an example of slowly-moving contact discontinuities adding another difficulty to numerical methods.

Results are given on figures 4.1 to 4.7.

The Godunov solver deals with shocks directly, it has a very low numerical diffusivity and is strictly conserving in mass, momentum and energy. In almost all cases, the results of the Godunov scheme and the van Leer scheme are comparable and accurate. In test 7 (see fig.4.7) we can notice that the Godunov scheme provides the most accurate solution. We can observe, for all tests that the LLF scheme provides solutions that are more diffusive.

On fig.4.1 and fig.4.2 we can observe that the most robust solutions are found either using the Godunov scheme or the HLLC scheme. We can say that the schemes are comparable, but the HLLC scheme is very sensitive to discontinuities. It fails firstly for the test 3, then for test 5 and 7 where it leads to the apparition of vacuum values. We get a vacuum value when the density is null and the total energy per unit becomes also zero (see [93] for more details). For test 4 the HLLC scheme fails to capture the internal energy. On fig.4.6 we can observe that the solution is far from the exact one. The AUSM$^+$−up scheme is also very sensitive and it fails to produce solutions to tests 4, 5 and 7. Fig.4.8 shows the results of all schemes for CFL number equals to 0.6. In this case the AUSM$^+$−up scheme provides the sharper solution, it resolves the contacts more accurately than all others schemes. Similar results are found for CFL number less than 0.6. For tests 4 and 5 the scheme fails even for small CFL number. The observed behaviour of the AUSM$^+$−up scheme is comparative with the one presented by Toro in [93] for the same tests, but using the AUSM or Liou Steffen scheme.

Each scheme behaviour is typical of a first order scheme. We can observe that the corners at the endpoints or the rarefaction wave are rounded, meaning that there are no spurious

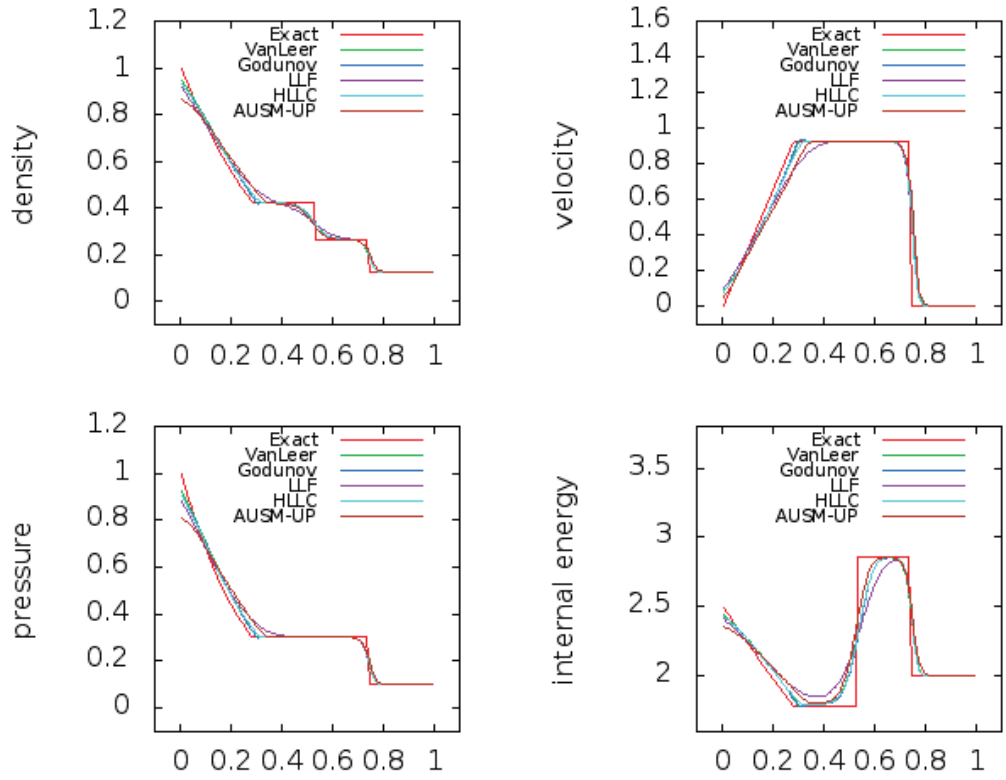


Figure 4.1: Test 1. Exact and approximated solutions for the Sod problem

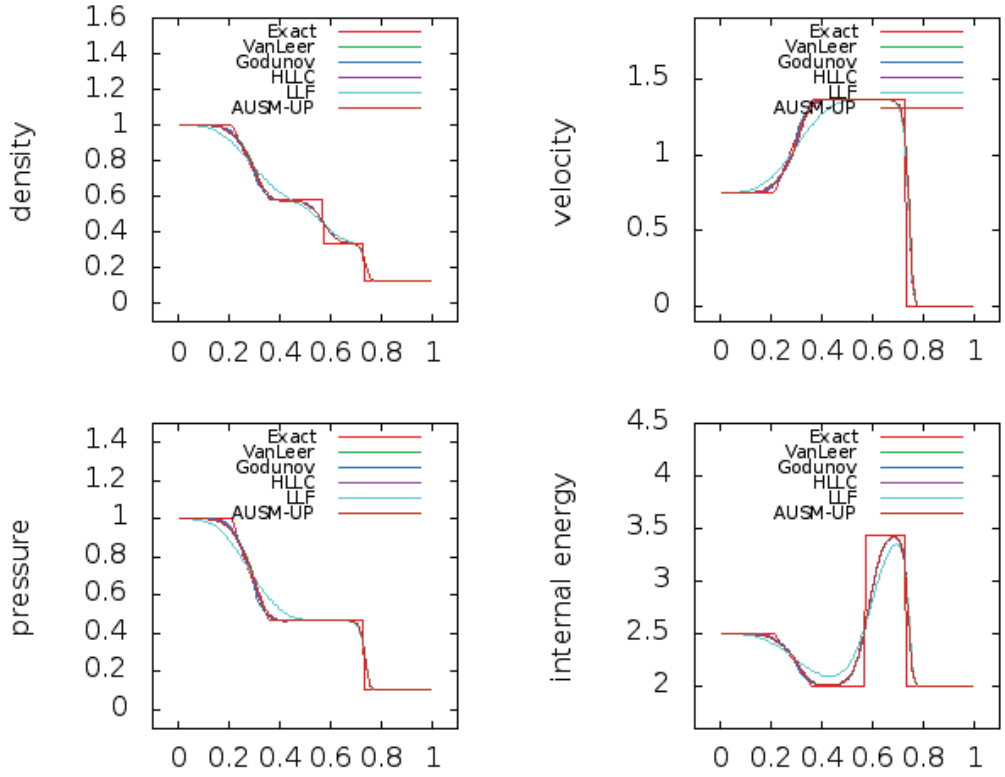


Figure 4.2: Test 2. Exact and approximated solutions for the modified Sod problem

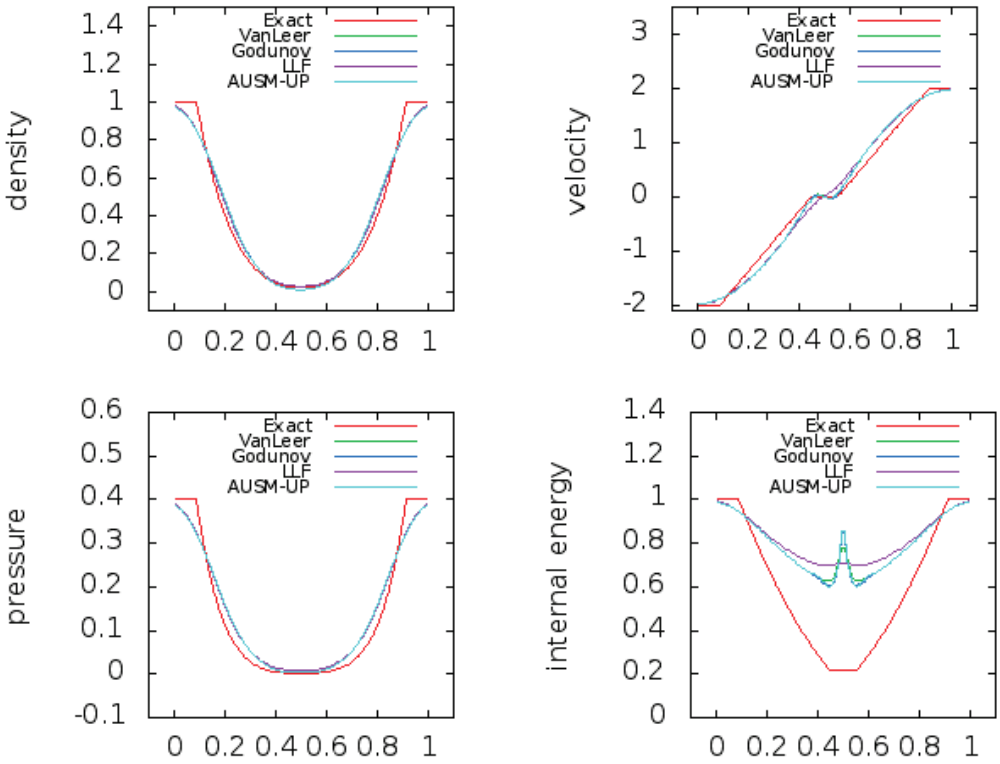


Figure 4.3: Test 3. *Exact and approximated solutions for the 123 problem*

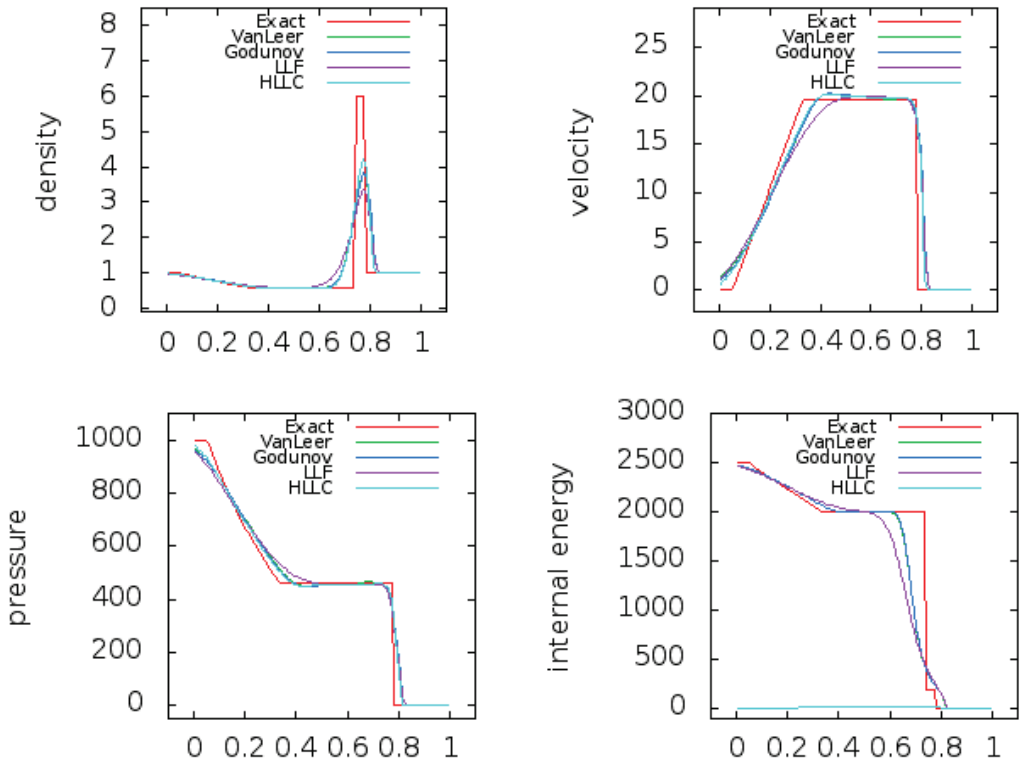


Figure 4.4: Exact and approximated solutions for test 4

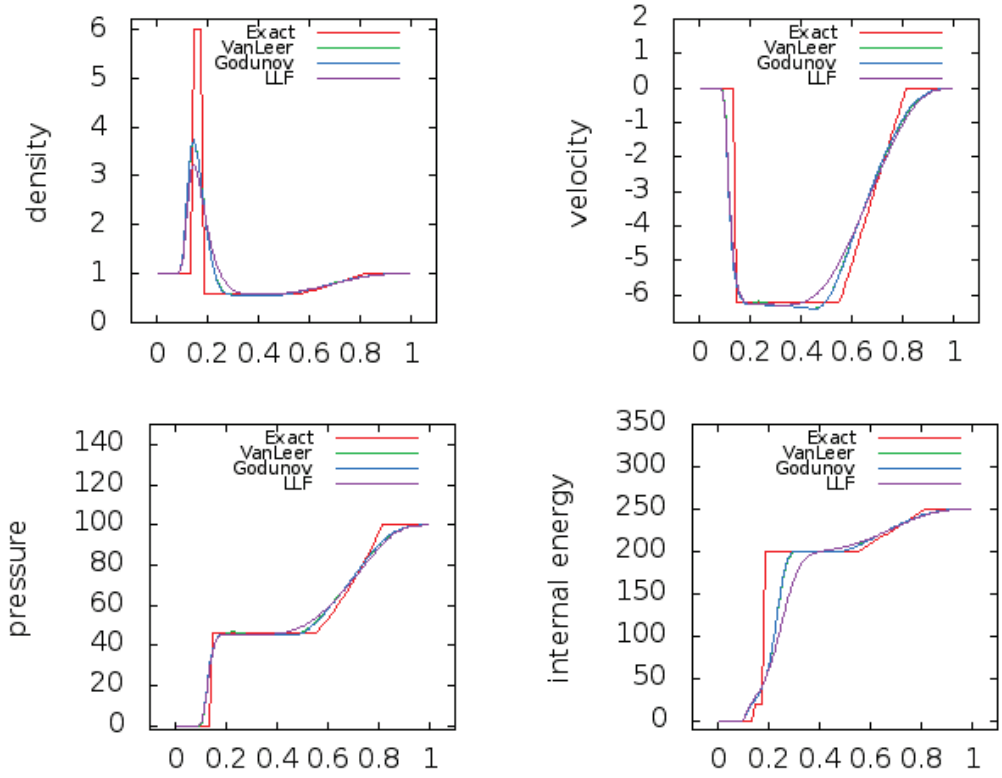


Figure 4.5: Exact and approximated solutions for test 5

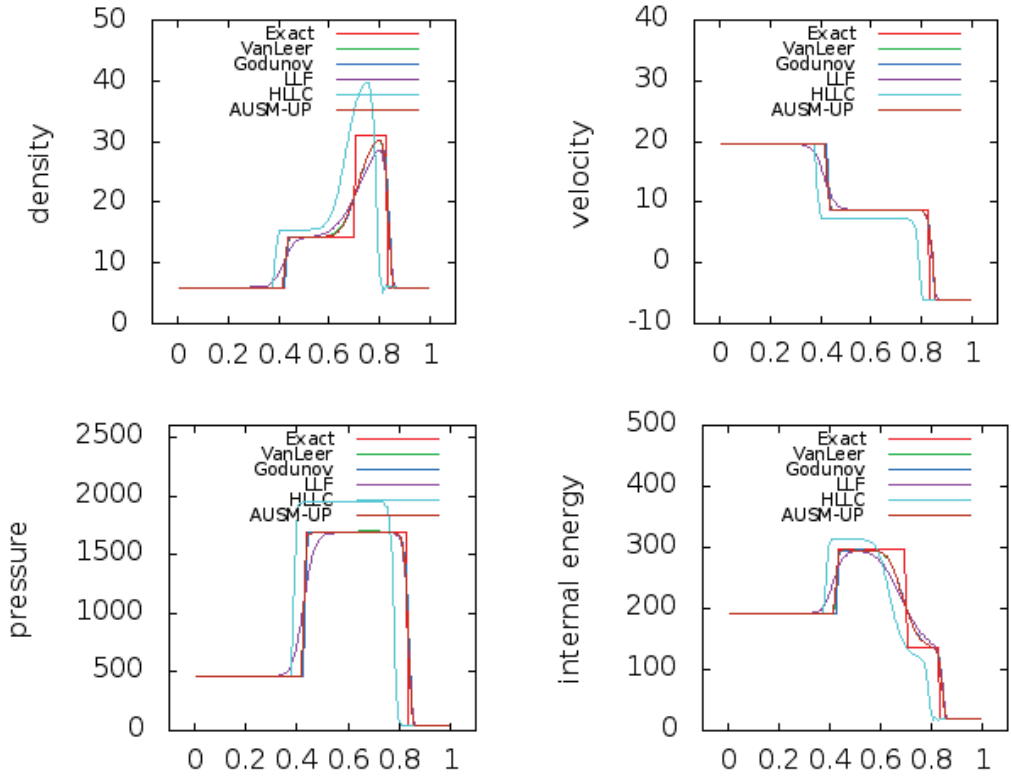


Figure 4.6: Exact and approximated solutions for test 6

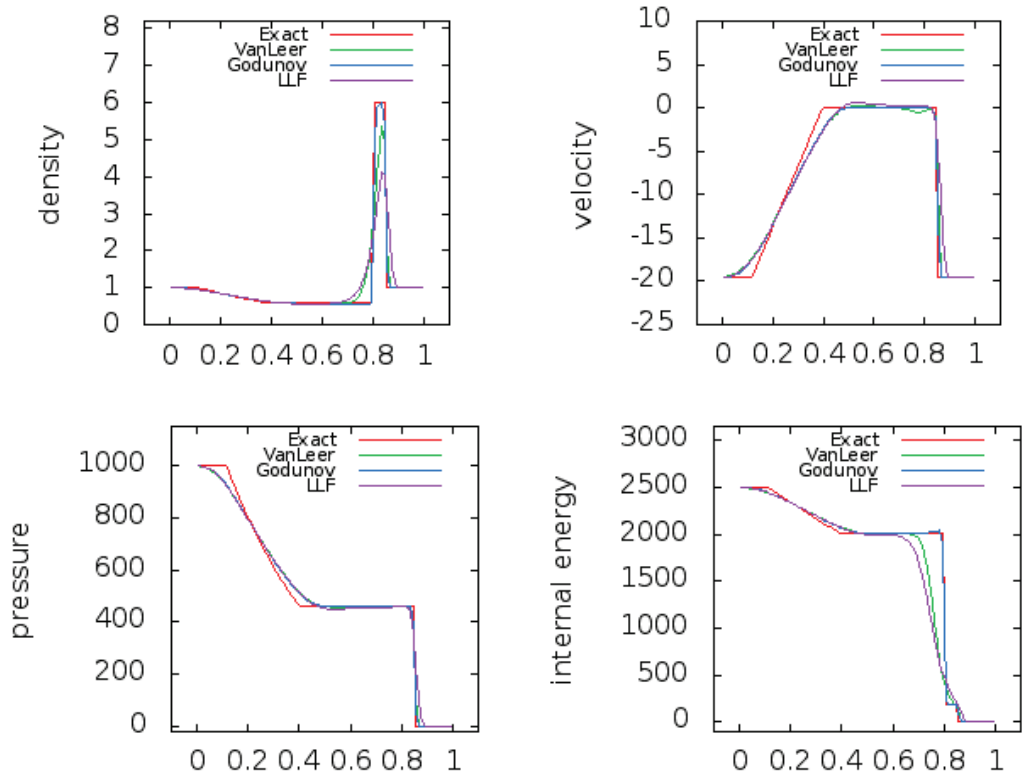


Figure 4.7: Exact and approximated solutions for test 7

oscillations in the vicinity of the shocks. It seems that the shocks are more sharply resolved when Godunov and van Leer schemes are used, the solution is more diffusive when splitting methods are chosen. On fig.4.3 we can observe large errors for the specific internal energy. It is explained by the close to zero values of density and pressure that are divided by their ratio when computing the internal energy. The test 3 proves that all schemes are sensitive to discontinuities and can easily fail to give a solution. The corresponding Mach number for the test 4 (see fig.4.4) equals 198. It is a very strong wave, but most of the schemes still manage to give a good solution.

In 1D cases, the Godunov and van Leer solvers prove to be very powerful. They tightly solve shocks and contact discontinuities with a high resolution. The spatial size or the order of accuracy should increase in order to get more exact solutions. We can already question the performance of the HLLC and AUSM$^+-$up for tests with discontinuities. Yet, we have observed that the AUSM$^+-$up gives accurate results for all tests where a solution is provided.

All presented results have been found solving the global system on one computational domain. No loss of accuracy has been noticed when the computational domain is divided into overlapping sub-domains with overlap of length half of stencil size (necessary for the use of ghost cells). The solution of the Euler/Navier–Stokes equations using the first order Euler Explicit scheme, or the second order Runge Kutta scheme computed on one global domain equals the one computed on a partitioned domain with Dirichlet type transmission conditions.

Most of the presented numerical procedure have proved to be stable and monotonic without leading to any spurious spatial oscillations. In addition, these procedures conserve mass and momentum locally. At this moment the schemes are limited by the CFL condition, and implicit temporal schemes that allow us to take larger time steps are studied in the next section.

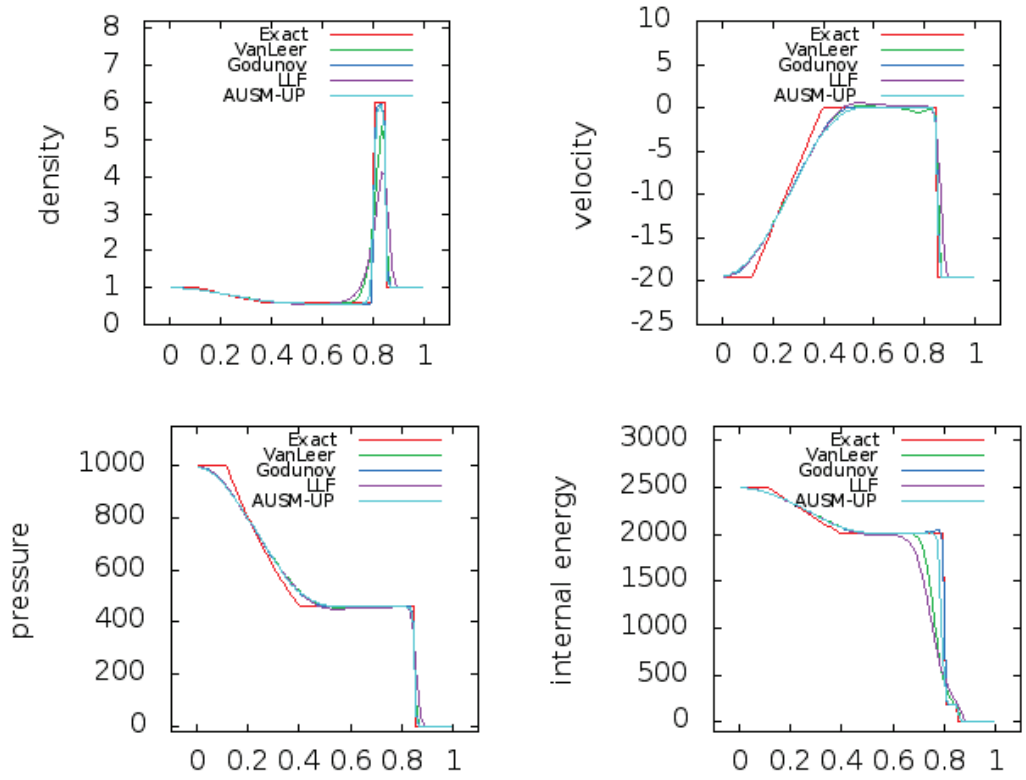


Figure 4.8: Exact and approximated solutions for test 7 with cfl= 0.6

4.1.2 2D Forward Facing Step

Shock capture can be very different between the one-dimensional and two-dimensional cases. A well-known shock capturing test in two dimensions is the Forward Facing Step case. It is the case of a flat-faced step in two-dimensional flow presented for the first time in 1968 by Ashley F. Emery [34]. It became known when Woodward and Colella publish, in 1984 a very detailed paper [98] on the analysis of numerical schemes and shock capturing where it is the main example. For a long period researchers have used this test to prove the efficiency of their schemes. We shall use it in order to validate the chosen Euler fluxes numerical schemes.

The geometry of the problem is presented on Fig.4.9. It is a wind tunnel of 1 length unit wide and 3 length units long containing a step. The step is located at 20% of the left end of the tunnel and 20% of the bottom end of the tunnel. The fluid is a perfect gas, $\gamma = 1.4$, the initial solution is supposed constant $(\rho, u, v, p)^t = (1.4, 3., 0., 1.)^t$ over the entire computational domain. The initial flow is in the supersonic regime of Mach number value 3. The tunnel is assumed to have infinite length or two-dimensional planar base flow and the flow is left to right oriented. In order to simulate this case, inflow boundary condition has been imposed at the left end and outflow boundary conditions at the right length. At the inflow the flow has the same characteristics as the initial one. The outflow boundary condition is of transmissive type has no effect on the computation, no spurious waves coming from the right are allowed to perturb the fluid inside the tunnel. Along the other walls the fluid is reflecting, the applied boundary conditions are reflective type.

The evolution of the solution leads to shock waves reflecting from the closed boundaries. A rarefaction wave is developed at the corner of the step and hence is a singular point of the flow. Woodward and Colella propose in [98] a particular treatment of this point to avoid possible numerical errors generated in its vicinity. This special treatment will not be applied in this

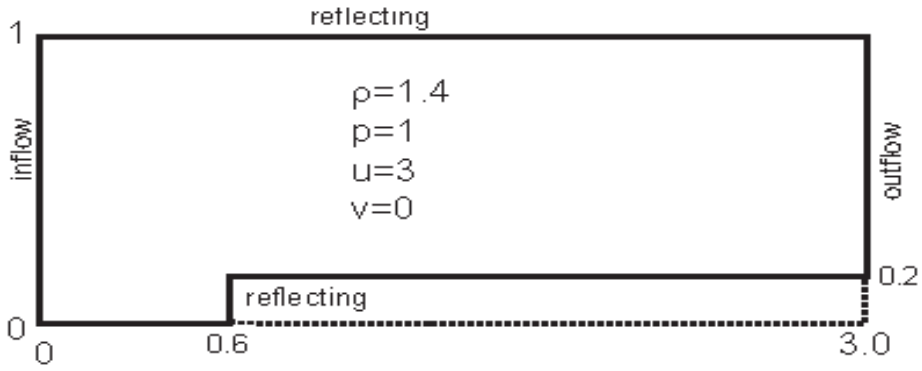


Figure 4.9: Wind tunnel with a step

work since it is a particular one and the goal of our tests is to validate the presented method in the view of future use on more complicated configurations.

The computational domain has been partitioned into five sub-domains as on Fig.4.10 and each sub-domain is solved in parallel on a different processor. The global domain size is $20 \times 50+4\times(20\times40)$ cells, the time step Δt is computed in order to satisfy a stability criteria of CFL number $cfl = 0.2$. In this section we present only results solving the Euler system of equations using a second order Runge Kutta method combined with a second order MUSCL scheme. We shall test the accuracy of the MUSCL scheme combined with each of the presented first order numerical schemes to compute the Euler fluxes. In section 1.2.3 we have presented different slope limiters for the MUSCL schemes. The Forward Facing Step solution develops strong discontinuities that allows to test the sensitivity of each limiter one of them. An important remark is that the computational domain is separated following the step, exactly where the shock will appear. Therefore, our numerical schemes need to converge not only when numerical errors are added by the singular point, but also need to manage the numerical errors due to the communication between the sub-domains.

The flow trajectory is guided by the reflecting walls. Until up to time 4 the flow is unsteady, two vortices appear just before the step and right after and a rarefaction wave is developed at the corner.

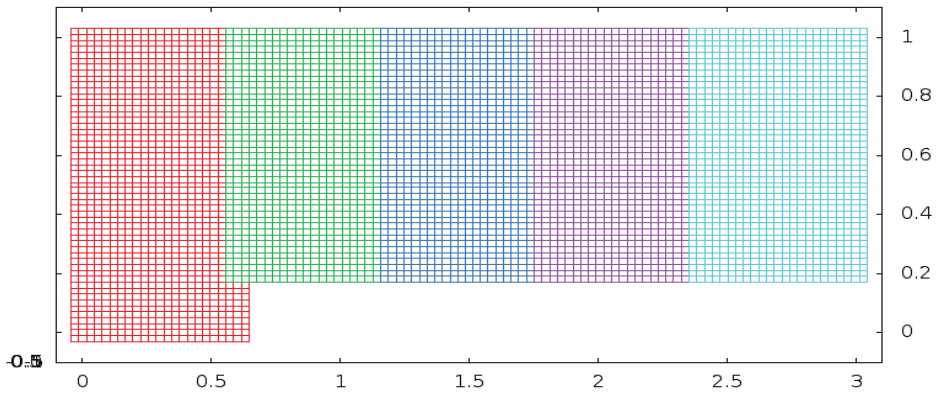


Figure 4.10: Forward Facing Step. The computational Domain split into fictitious overlapping sub-domains (overlap of half stencil size).

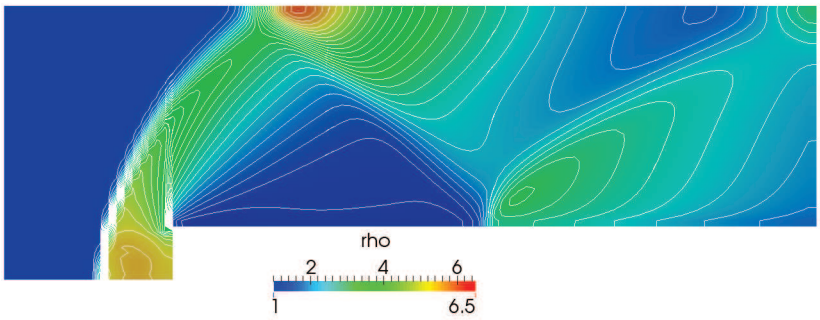
At time 4 we can already see the stabilised trajectory of the flow and by time 12 the flow

becomes steady. We focus here on the unsteady flow where we can observe the shock evolution. We show results for $1, 2, 3$, and 4 units of time. The Euler fluxes are first computed using a first order scheme. On each figure we outline thirty equally spaced contours for 1 to 6.5 that will help us identifying the trajectory of the flow, the vortices and the shocks. The choice is different from the one of Woodward and Colella who decided to adapt the distance between the contours for each solution and so to always have the same number of contours. The choice to fix the values of the contours for all computed solutions was established in order to observe qualitatively the amount of artificial diffusion added by each numerical scheme. We shall still be able to compare the solutions with those presented in [98].

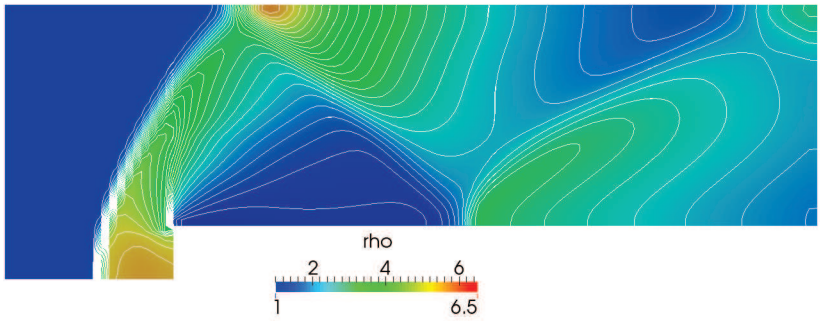
The solutions seen in fig.4.11i to fig.4.8iv are computed with the first order scheme. On figures 4.11i, 4.10ii, 4.9iii and 4.8iv one can see the approximated solutions at 4 units of time using the Godunov, the van Leer, the AUSM$^+$−up, respectively the LLF first order numerical schemes to compute the Euler fluxes. We are comparing the general shape and the position of the shocks. We can observe the evolution of the thinness of the shocks, when the flow is nearly steady the shocks are very thin. The explanation comes from the fact that the jumps in the shocks are better resolved and contours are focused at the same position.

We consider as reference the solution found with a fine uniform grid of size $100 \times 250 + 4 \times (100 \times 200)$ cells computed using a MUSCL scheme with the Van Albada type slope limiter. On fig.4.6i and fig.4.5ii we can observe the solutions on a fine mesh using the AUSM$^+$−up scheme and the LLF scheme to compute the Euler fluxes. They are the only schemes able to give a solution with a very small space step that introduce very large numerical errors. As before, for all presented results the LLF scheme manages to give a solution, but it diffuses more than all the others schemes. We can observe that the solution found with the AUSM$^+$−up scheme captures very well all features of the flow field and that the results compare very closely with those presented by Woodward and Colella in [98]. The AUSM$^+$−up scheme, on the fine mesh shows the evidence of vortex sheet roll up. These details can be seen using others schemes only for a more refined mesh.

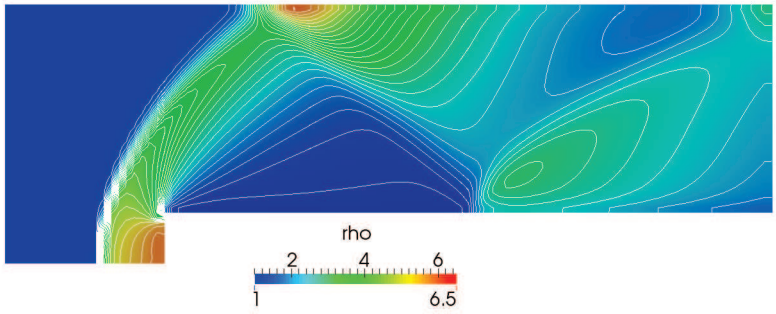
We continue by computing the Euler fluxes using a second order MUSCL scheme with a Van Albada and MinMod slope limiter combined with the Godunov, the van Leer, the AUSM$^+$−up, the LLF and the HLLC scheme.



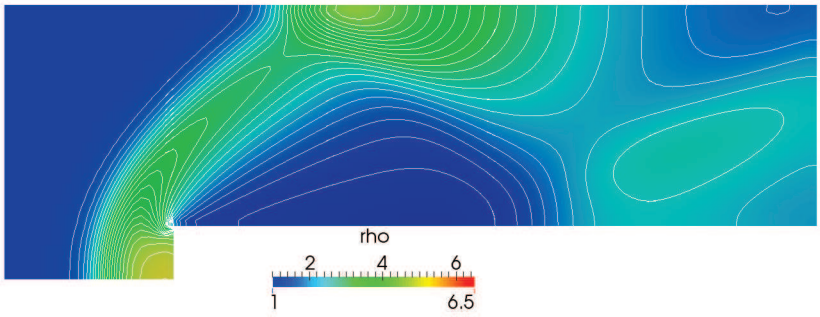
(i) Approximated solution for 4 units of time obtained with a first order Godunov flux scheme



(ii) Approximated solution for 4 units of time obtained with a first order van Leer flux scheme

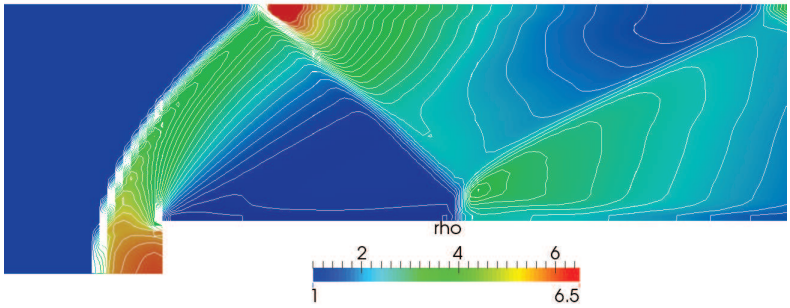


(iii) Approximated solution for 4 units of time obtained with a first order AUSM$^{+}-$up flux scheme

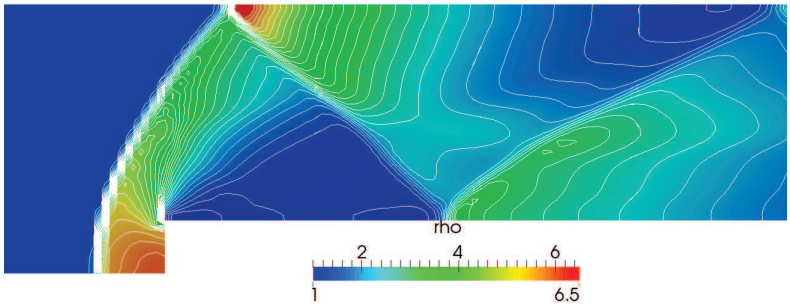


(iv) Approximated solution for 4 units of time obtained with a first order LLF flux scheme

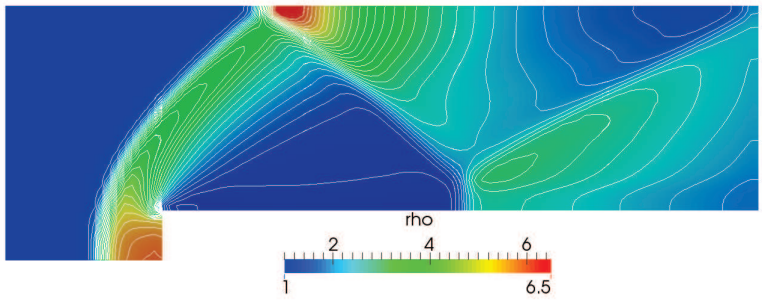
Figure 4.8: Approximated solution obtained with a first order scheme



(i) Approximated solution for 4 units of time obtained with a MUSCL scheme with MINMOD limiter and first order AUSM^{+}−up flux scheme

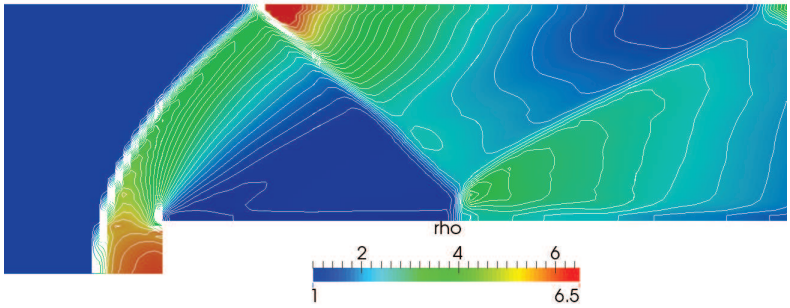


(ii) Approximated solution for 4 units of time obtained with a MUSCL scheme with MINMOD limiter and first order van Leer flux scheme

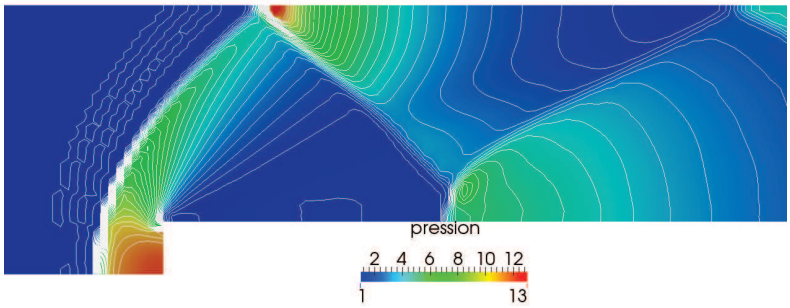


(iii) Approximated solution for 4 units of time obtained with a MUSCL scheme with MINMOD limiter and first order LLF flux scheme

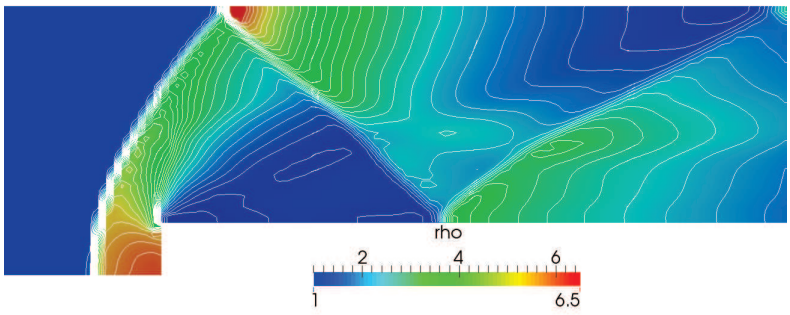
Figure 4.7: Approximated solution obtained with a second order MUSCL scheme with MINMOD limiter



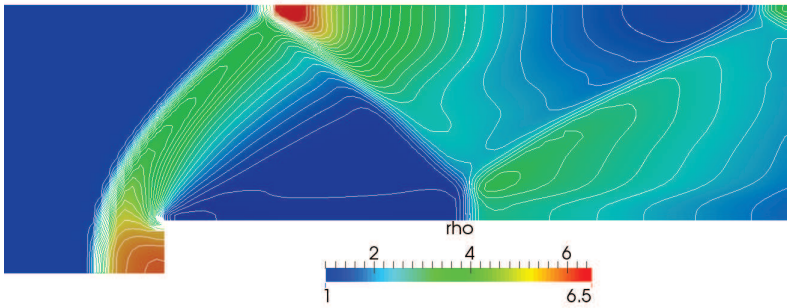
(i) Approximated solution for 4 units of time obtained with a MUSCL scheme with Van Albada limiter and first order AUSM$^+$−up flux scheme



(ii) Approximated solution for 5 units of time obtained with a MUSCL scheme with Van Albada limiter and first order AUSM$^+$−up flux scheme

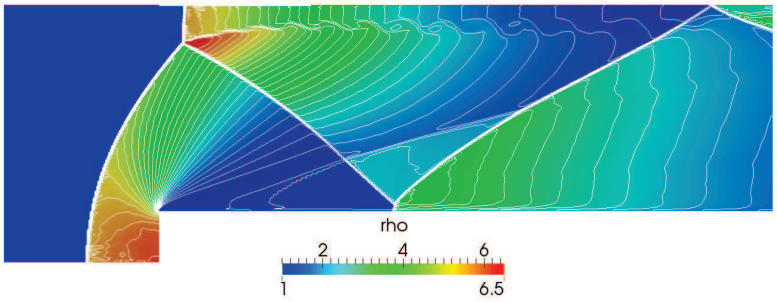


(iii) Approximated solution for 4 units of time obtained with a MUSCL scheme with Van Albada limiter and first order van Leer flux scheme

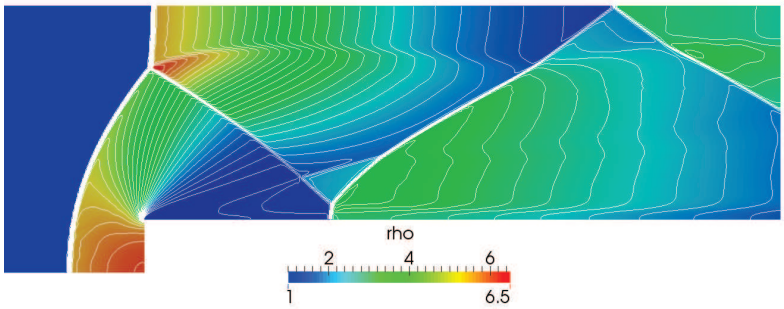


(iv) Approximated solution for 4 units of time obtained with a MUSCL scheme with Van Albada limiter and first order LLF flux scheme

Figure 4.5: Approximated solution obtained with a second order MUSCL scheme with Van Albada limiter



(i) Fine grid approximated solution for 4 units of time obtained with a MUSCL scheme with Van Albada slope limiter and first order AUSM$^{+}-$up flux scheme



(ii) Fine grid approximated solution for 4 units of time obtained with a MUSCL scheme with Van Albada slope limiter and first order LLF flux scheme

Figure 4.5: Fine grid approximated solution obtained with a second order MUSCL scheme with Van Albada limiter

4.2 Applications of Domain Decomposition techniques

In this section, we present and analyse numerical results with parallel techniques in four different test cases.

First, the Euler system of equations is solved for a simple configuration on CPU and we compare the results with those found using exactly the same second order algorithm, but on GPUs (see section 3.2.3 for details on GPU implementation).

Secondly, we focus on MIMD architectures. Performance of the different parallel computing strategies (using OpenMP, MPI) are compared: a) on the inviscid and viscous motion of a 2D isolated vortex in an uniform free-stream, b) on the case of the sound generation in a 2D mixing layer, and c) the vortex shedding from rectangles. Space discretisation is achieved with finite volumes on Cartesian non-uniform grids. The sub-domains overlap region has either half stencil size ($\theta = 1$ for first order schemes and $\theta = 2$ for second order schemes) or half stencil size plus one. We use a second order projection method to exchange data in time and in space. All implicit algorithms are second order in time and space. Common to all schemes is the second order implicit BDF scheme. Whenever, in the following subsections, we talk about the Newton scheme, we refer to the Newton-Partitioning algorithm presented in section parallel partitioning implicit scheme. The linear solver inside the Newton process is solved in parallel inside sub-domains. The cost is measured by total computational time or in the total number of iterations, necessary to obtain a given level of accuracy.

4.2.1 GPU versus CPU

The aim of this test is to show that the performance of the presented methods can still be improved. The methods can be accelerated using GPUs. GPUs can be used to solve a global problem or a local one using a massive parallel architecture. We shall only solve a mono-domain Euler system of equations with an explicit second order Runge Kutta method. The Euler fluxes are computed using the MUSCL second order scheme with a Van Albada limiter. The first order fluxes are computed with AUSM$^+$-up Advection Upstream Splitting Method) scheme. The advantage of the AUSM$^+$-up, developed in [69], is that it was conceived to be uniformly valid for all speed regimes. At this point, it is the only scheme that we have implemented on the GPU.

We start by solving the global problem on a GPU (NVIDIA Corporation GF110 [Geforce GTX 580] Compute Capability 2.0) with CUDA, launched from a CPU, and compare its computational cost with one running on a CPU (7.8 GB, 2 Cores at 3.33GHz) with OpenMP.

The computational domain, on fig.4.6, is a tube of size $\Omega = [0., 10.] \times [0., 20.]$. We consider that the tube is infinite in the $x-$direction (right outflow boundary condition). At the left side, we impose an inflow velocity, $u = \frac{\pi}{5}\cos(y-5), v = 0$, at each time step. We consider that the initial density and pressure are constants of unit size.

We impose a CFL stability condition equal to 0.5 and recompute the time step at each instant. We stop the simulation after $T = 100s$. We increase the computational domain size and show the ratio of the computation on a CPU with 2 cores (OpenMP with the computation on CPU with one core combined with a GPU, on table 4.2. We show the total computational cost and the cost of different steps of computation: the time step computation via a minimum reduction, the computation of numerical convective fluxes (MUSCL + Van Albada + AUSM$^+$-up), the updates of the boundary and of the solution. We get different performance depending on the step. It is due to the fine granularity and optimisation of each step.

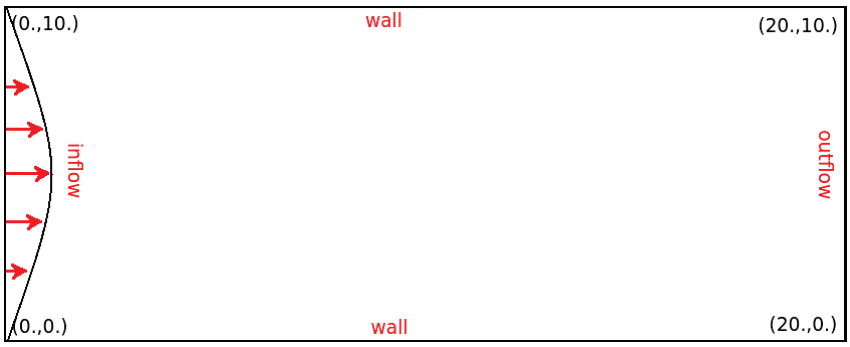


Figure 4.6: Initial domain, Ω: physical size and boundaries.

Table 4.2: (CPU 2 Cores + OPENMP computational cost) / (CPU 1 Core + GPU computational cost) for different steps of computation.

Grid Size	Time Step computation	2D Fluxes	Update Step	Bnd Update	Total
130x130	43.08	1.63	8.62	0.31	3.72
260x260	109.26	1.71	15.90	1.58	4.65
525x525	164.83	2.81	40.37	1.38	6.88
1050x1050	392.72	2.58	321.21	2.39	7.80

As can be seen there is a definite gain to be obtained on the CPU-GPU configuration. The gain increases with the computational domain size. Computation on both configurations are quite scalable. GPU code is portable on any NVIDIA GPUs using CUDA programming model, though, it should be noted that performance on GPUs vary a great deal depending on the GPU specifications. We remark that [Geforce GTX 580] is the first generation of GPUs capable to compute 3D computations. A recent NVIDIA GPU, such as Kepler or Maxwell, could improve the performance. Our implementation strategy was to intensively optimise codes for both parallel or non-parallel solvers. Although, we consider that the previous results are correct, we admit that comparing results with equivalent solvers is not the best strategy. GPU and CPU optimisation strategies are not always similar.

4.2.2 Exact solution for Euler equations: 2D isentropic vortex

The second 2D test presented in this chapter is the Isentropic Vortex Evolution based on Yee's paper [99] in 1999, for which we will study the numerical diffusion of the presented schemes. This test is a particularly interesting case for the Euler equations since it provides an exact solution.

The computational domain is a square of size $[-5.,5.]\times[-5.,5.]$. In the initial configuration a vortex is placed in the middle of the domain. The mean flow density, velocity and pressure $(\rho_\infty, u_\infty, v_\infty, p_\infty)$ are considered to be free stream and the fluid a perfect gas, $\gamma = 1.4$, $\frac{p}{\rho^\gamma} = 1$. The values of the density and the pressure are fixed, $\rho_\infty = p_\infty = 1$. The vortex is convected in a specified direction depending on the values of u_∞ and v_∞. In this particular example, the results are obtained for a convected vortex in a diagonal direction $(u_\infty, v_\infty) = (1,1)$ with no shock waves or turbulence. The initial condition equals the mean flow field plus an isentropic

vortex with no perturbation in entropy:

$$\begin{aligned}
\rho &= (T_\infty + \delta T)^{\frac{1}{\gamma-1}} &&= (1 - \tfrac{(\gamma-1)\beta^2}{8\gamma\pi} e^{1-(x^2+y^2)})^{\frac{1}{\gamma-1}} \\
\rho u &= \rho(u_\infty + \delta u) &&= \rho(1 - \tfrac{\beta}{2\pi} e^{\frac{1-(x^2+y^2)}{2}}) \\
\rho v &= \rho(v_\infty + \delta v) &&= \rho(1 + \tfrac{\beta}{2\pi} e^{\frac{1-(x^2+y^2)}{2}}) \\
p &= \rho^\gamma \\
e &= \tfrac{p}{\gamma-1} + \tfrac{1}{2}\rho(u^2+v^2)
\end{aligned} \quad ,$$

where β is the vortex strength. We suppose that the domain has infinite length and impose periodic boundary conditions in both x and y direction. At the end of each cycle that lasts 10s the vortex equals the initial solution.

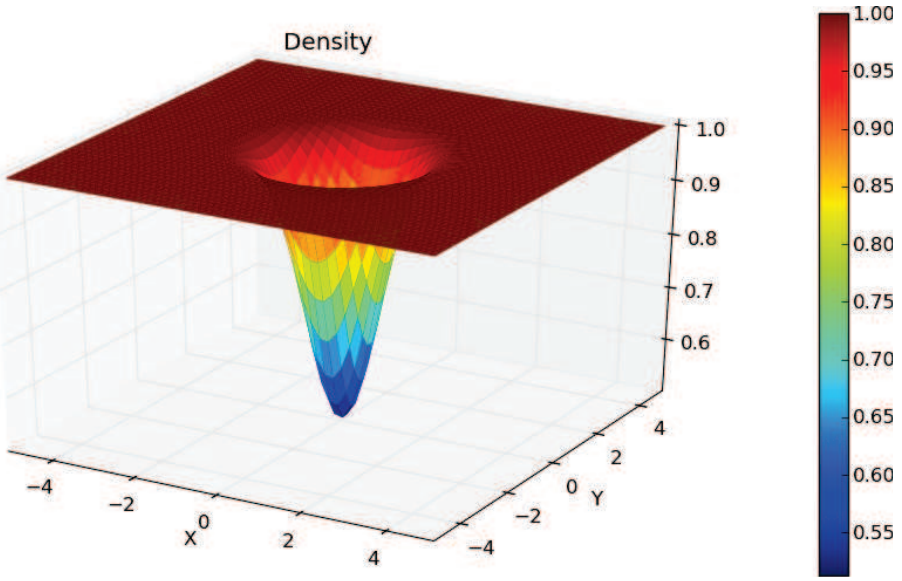


Figure 4.7: Initial solution of the 2D Isentropic Vortex

The following accuracy study and estimation of computational costs are results on a convective vortex with $(u_\infty, v_\infty) = (1, 1)$. The overlap equals half the stencil size ($\theta = 2$). We use periodic boundary conditions and Dirichlet type coupling conditions. At the end of one cycle, that lasts 10s, the vortex equals the initial solution.

Accuracy study

Let us begin with an accuracy study of the Euler equations computing L^2 and L^∞ slopes of relative errors. First, let us fix the number of fictitious overlapping sub-domains to 2×2 and a common time step. We increase the global number of space cells from 40×40 cells to 60×60 cells and 80×80 cells. When a common time step is used, the adaptive SWR methods are identical to the classical SWR method. We consider that we have converged when we reach an error less than a tolerance equal to $1.e-6$ for both Newton and Schwarz stopping criteria. The Schwarz stopping criteria is a L2 relative error of variations between overlap regions. We suppose that the overlap region of two neighbour sub-domains should be coincident.

As can be seen in Fig.4.8, all presented methods are second order in time and close to second order in space, depending on the Van Albada limiter chosen for the MUSCL scheme. Velocity, pressure and energy errors behave similarly for all presented methods. The method denoted as Newton in Fig.4.8 is the Newton-Schwarz method using only one Schwarz iteration, it only has order one accuracy, it converges to a false solution showing that the Schwarz iterative procedure is a good preconditioner for our scheme.

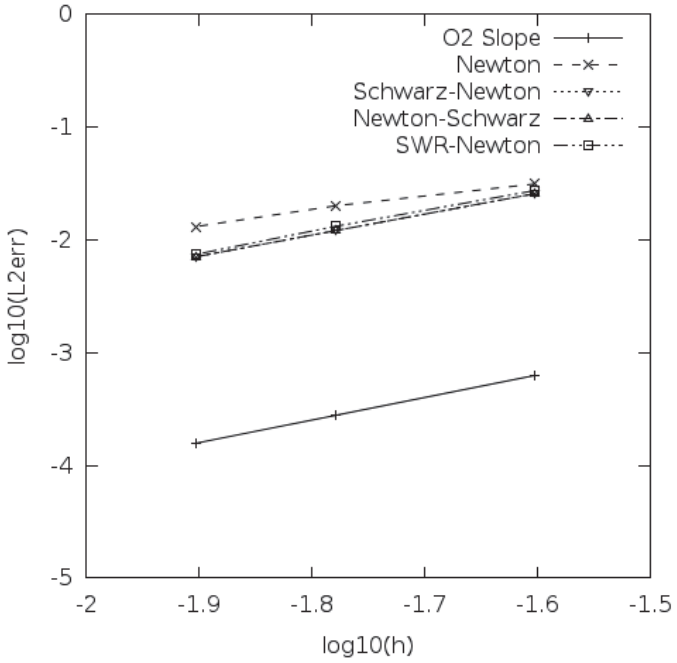


Figure 4.8: L^2 error over density field

All implicit schemes are stable for large cfl number. A more relevant result to evaluate the cost is the number of times the local linear system has to be solved for varying CFL number and varying number of sub-domains.

Computational cost

The number of local linear solves is given by the product between the number of Newton iterations and the number of Schwarz iterations and it is a good measure of the computational cost. This cost is a linear function of the number of cells. On Table.4.3 are shown the average number of local linear solves per time step for the Newton-Schwarz method, the Schwarz-Newton method and for the SWR-Newton (SWR) method for an increasing number of sub-domains with a fixed number of reduced size cells (weak scalability). The sub-domain size is fixed to 20×20 points and the cfl number has the value 0.5. For the SWR-Newton scheme, we choose δt the same time step on each sub-domain and $\Delta T = 5\delta t$ the time window. The Newton stopping tolerance is set to $1e-6$. The Schwarz convergence tolerance is varying as shown on Table.4.3.

We solve implicitly, the system of Euler or Navier–Stokes equations using a Newton based method and an approximated $\mathcal{L}+\mathcal{D}+\mathcal{U}$ method to solve the resulting linear system. The size of the linear system is fixed inside each sub-domain. Consequently, we can use the linear solver as a first unit of measuring the weak scalability.

When varying the Schwarz convergence tolerance we observe an important decrease of the number of linear solves when the SWR method is used to solve the system of equations. Between the Schwarz tolerance set at $1.e-6$ and $1.e-2$ we gain a factor more than 3 when the SWR method is used, but only a factor less than 2 when the Newton-Schwarz or the Schwarz-Newton method is used. We can observe that, for the same tolerance, the number of linear solvers is not constant thus the variation can be explained by the different space steps when increasing the number of sub-domains.

On Table.4.3, the communication between 4 processors is done, using the SWR-Newton scheme on an average of 18.6 times per time window. In order to reach the same time window the Schwarz-Newton scheme communicates in average 35.45 times and the Newton-Schwarz scheme communicates in average 250 times.

Table 4.3: Weak scaling of the schemes. Computational costs

		Schwarz tol = 1.e-6			Schwarz tol= 1.e-3			Schwarz tol = 1.e-2		
Scheme	Iterations\Sub-domains	4	9	16	4	9	16	4	9	16
Newton-Schwarz	Newton iterations	8.48	8.11	6.94	8.48	8.11	6.93	8.48	8.11	9.29
Newton-Schwarz	Schwarz iterations	5.89	6.28	6.41	5.26	5.70	5.78	4.71	5.00	5.66
Newton-Schwarz	Linear Solvers	50.00	51.02	44.51	44.64	46.33	40.15	39.89	40.63	52.62
Schwarz-Newton	Newton iterations	3.07	2.72	2.28	5.00	4.18	3.28	5.93	4.90	5.31
Schwarz-Newton	Schwarz iterations	7.09	7.49	7.15	3.55	4.0	4.0	2.46	3.0	3.0
Schwarz-Newton	Linear Solvers	21.79	20.40	16.31	17.78	16.75	13.15	14.63	14.72	15.93
SWR	Newton iterations	7.54	6.62	5.41	7.52	6.61	5.37	7.49	6.56	7.13
SWR	Schwarz iterations	18.6	14.28	19.25	7.08	7.98	7.44	4.36	4.77	5.66
SWR	Linear Solvers	140.6	95.3	104.28	53.28	52.78	39.99	32.75	31.43	40.40

On Table. 4.4 are shown the average number of local linear solves per time step for all presented methods for an increasing number of sub-domains with a fixed global domain size (strong scalability) of 120x120 cells.

Table 4.4: Strong scaling of the schemes. Computational costs

		Schwarz tol = 1.e-6			Schwarz tol= 1.e-3			Schwarz tol = 1.e-2		
Scheme	Iterations\Sub-domains	4	9	16	4	9	16	4	9	16
Newton-Schwarz	Newton iterations	6.04	6.05	6.06	6.04	6.05	6.06	6.04	6.05	6.06
Newton-Schwarz	Schwarz iterations	6.27	6.56	6.73	5.77	5.89	6.23	4.80	5.01	5.22
Newton-Schwarz	Linear Solvers	37.94	39.77	40.85	34.95	35.67	37.83	29.07	30.35	31.72
Schwarz-Newton	Newton iterations	2.57	2.33	2.17	3.43	3.55	3.33	5.08	4.64	4.10
Schwarz-Newton	Schwarz iterations	6.78	7.04	7.31	4.42	3.67	3.59	2.03	2.0	2.30
Schwarz-Newton	Linear Solvers	17.46	16.43	15.90	14.75	12.64	11.98	10.35	9.28	9.46
SWR	Newton iterations	6.20	5.57	5.31	6.18	5.54	5.27	6.12	5.48	5.21
SWR	Schwarz iterations	17.28	18.60	19.01	10.57	8.81	7.23	3.70	4.00	4.27
SWR	Linear Solvers	107.24	103.68	103.68	65.27	49.77	38.12	22.69	21.97	22.28

These two tables show the good strong and weak scalability of all considered methods. It should be added that for large data problems, the communications between processors may be costly, and this cost increases when GPUs are associated with CPUs. When GPUs computation is involved the communication may be more expensive then the local computation.

The SWR method is ideal for clusters with high latencies.

One aim of this study is to find a method that converges independently of the number of sub-domains for the same number of computations. In this particular case, all methods behave well, the number of local linear solves even decreases compared to the increasing number of sub-domains. Yet, the number of Schwarz iterations is large compared to the number of Newton iterations. This is probably due to the poor Dirichlet type coupling conditions that we use and could be corrected using higher order coupling conditions such as unsteady Robin type conditions. However, our strategy to impose Robin type transmission conditions, presented in section 2.5, is unstable for overlap of half the stencil size.

The adaptive time step SWR method converges to the solution in exactly the same way as the fixed time step SWR method. The gain of the SWR method comes from the improved stability of the scheme since the time step is recomputed at each iteration thus less communication between the sub-domains as it appears that when the coupling conditions are improved, larger times steps are usually needed. This also leads to less CPU memory when fewer coupling conditions need to be stored.

Study of transmission conditions for overlap of size $\theta = 3$

In the previous paragraph we have seen that, in order to achieve expected results, one does not need to fully converge in the Schwarz process. Moreover, the implicit solver is an Inexact Newton which we do not expect to fully converge. We propose then to check the influence of the Dirichlet (Dirichlet type transmission conditions for both convective and diffusive fluxes) and Robin (Robin type transmission conditions for both convective and diffusive fluxes) transmissions conditions in the case of overlapping sub-domains with overlap $\theta = 3$. We compare results found on different configurations and number of equal sized sub-domains (two sub-domains and four sub-domains) with Dirichlet and Robin type transmission conditions. We vary the Robin coefficient which is supposed global (equal Robin coefficient in each sub-domain).

a) Two sub-domains

First, we split the global domain $\Omega = [-5, 5] \times [-5, 5]$ in two sub-domains of equal size 1800 cells, and distribution: $\Omega_1 = [-5, -2] \times [-5., 5.]$ of size 30×60 cells, and $\Omega_2 = [0., 5.] \times [-2., 5.]$ of size 30×60 cells. We ask a precision of $1e-5$ in the Newton convergence test and one of $1e-2$ for the Schwarz process stopping criteria. Also, we replace the previous stopping criteria, based on the relative error norm on the variation between overlapping cells with the one presented in the section 2.6 based on the variation error of the extremes cells surrounding the meshes (excepting the ghost cells).

The results for the classical schemes (Newton-Partitioning, Newton-Schwarz and Schwarz-Newton) are shown on fig.4.9. We observe that the results found with the Robin type condition are similar to those found with Dirichlet type condition and this for any Robin coefficient. Although we have established that the classical partitioning Newton method converges to a different solution of lower accuracy, we show its results since it is the cheaper one and can be seen as the best scenario in terms of minimum number of linear solves.

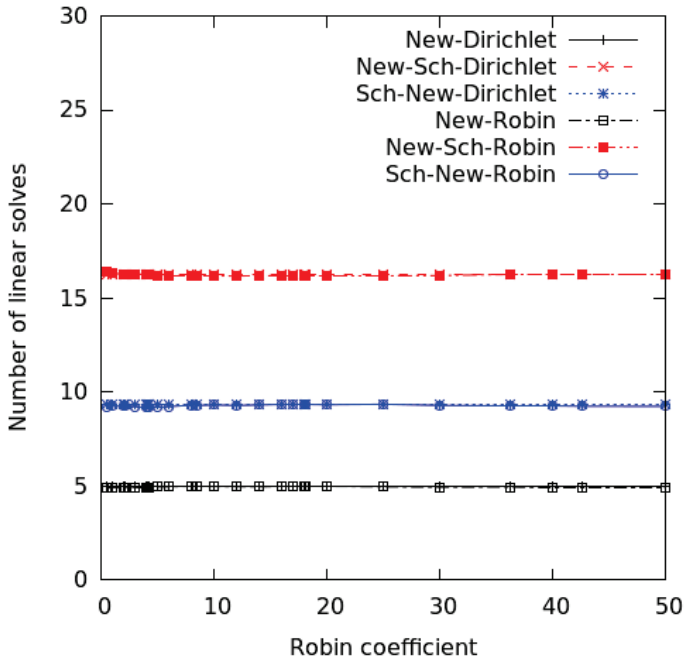


Figure 4.9: Average number of linear solves per iterations for 2 non-matching sub-domains.

On fig.4.12 we can see the results found with the SWR methods. 0 Level SWR represents the SWR method

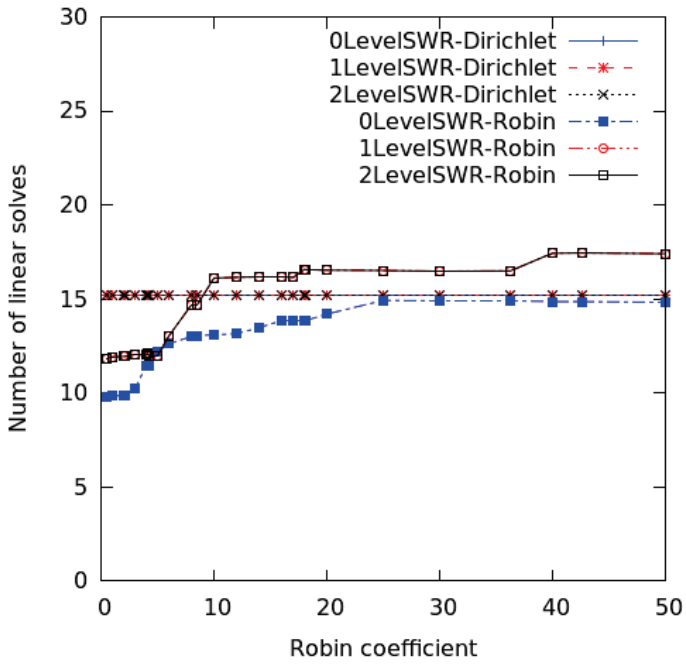


Figure 4.10: Average number of linear solves per iterations for 2 non-matching sub-domains. 0LevelSWR represents the classical SWR method with fixed discretisation inside a time window. 1LevelSWR is the one level adaptivity SWR method: the time steps inside a window are locally computed at the first iteration. 2LevelSWR is the SWR method with two levels of adaptivity: the time steps are locally computed at each time iterations.

For the SWR methods, the results can be seen on fig.4.12 and zoomed on fig.4.13. All three presented methods: the classical SWR with fixed time step, the 1 level of adaptivity SWR and the 2 level of adaptivity SWR (algorithms described in section 2.3) give similar results when the Dirichlet transmission condition is used. When a Robin type transmissive condition is used, the results are improved for all methods for Robin coefficients smaller than $\frac{1}{\min(\Delta x,\Delta y)}$ (equal to 10 in this case). The classical SWR with fixed time step (the time step is fixed to the smallest one and it is the same in each sub-domain) shows better results when Robin type transmission conditions are imposed, for any Robin coefficient.

b) Four sub-domains

We split the global domain $\Omega = [-5, 5] \times [-5, 5]$ in four sub-domains of equal size 1800 cells, but unequal distribution: $\Omega_{11} = [-5, -2] \times [-5, 0]$ of size 30×60 cells, $\Omega_{12} = [-2, 5] \times [-5, 0]$ of size 45×40 cells, $\Omega_{21} = [-5, 0] \times [-5, 0]$ of size 30×60 cells, $\Omega_{11} = [0, 5] \times [-5, 0]$ of size 45×40 cells. We ask a precision of $1e-5$ in the Newton convergence test and one of $1e-2$ for the Schwarz process stopping criteria. The stopping criteria is based on the variation error of the extremes cells surrounding the meshes (excepting the ghost cells).

The results for the classical schemes are shown on fig.4.11. We observe that the results found with the Robin type condition are at most as good as the results found with Dirichlet type condition and this for any Robin coefficient.

For the SWR methods, the results can be seen on fig.4.12 and zoomed on fig.4.13. All three SWR methods: the classical SWR with fixed time step, the 1 level of adaptivity SWR and the 2 levels of adaptivity SWR give similar results when the Dirichlet transmission condition is used. When a Robin type transmissive condition is used, the results are improved for all methods for Robin coefficients smaller than $\frac{1}{\min(\Delta x,\Delta y)}$ (equal to 10). For the classical SWR with fixed

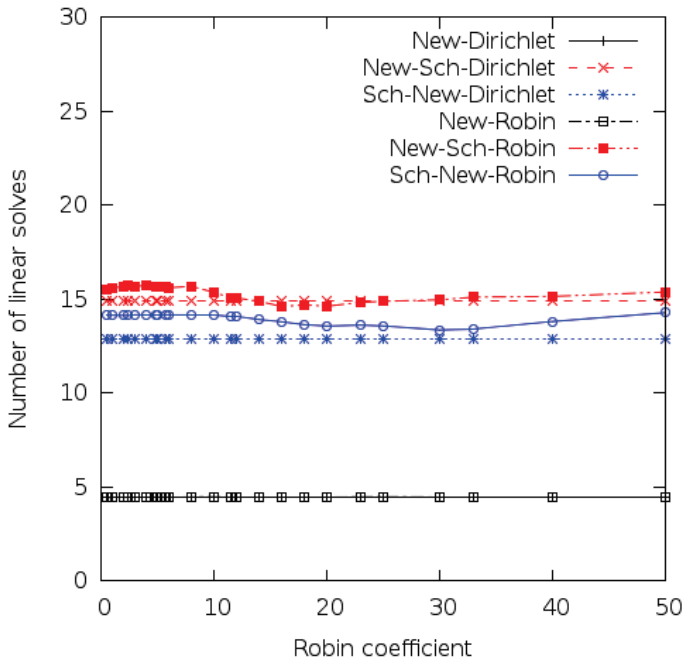


Figure 4.11: Average number of linear solves per iterations for 4 non-matching sub-domains.

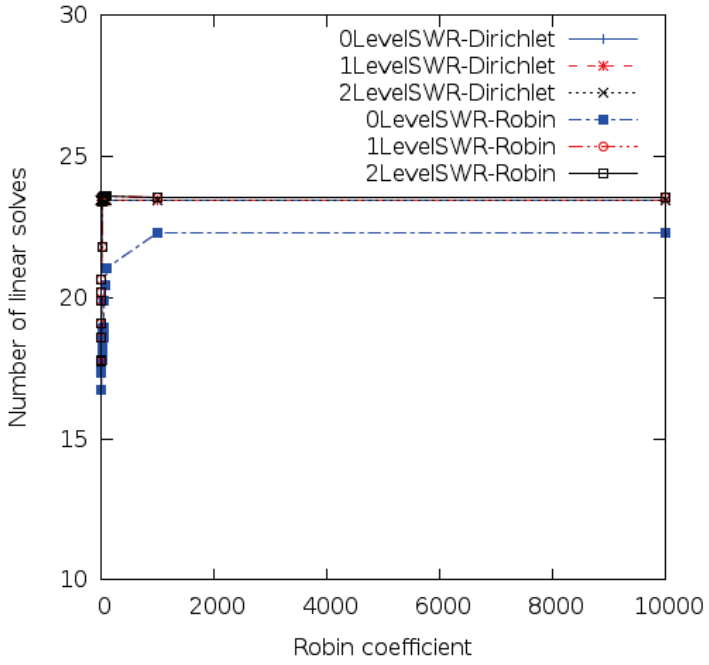


Figure 4.12: Average number of linear solves per iterations for 4 non-matching sub-domains.

time step (the time step is fixed to the smallest one and it is the same in each sub-domain), the results are improved for any Robin coefficient.

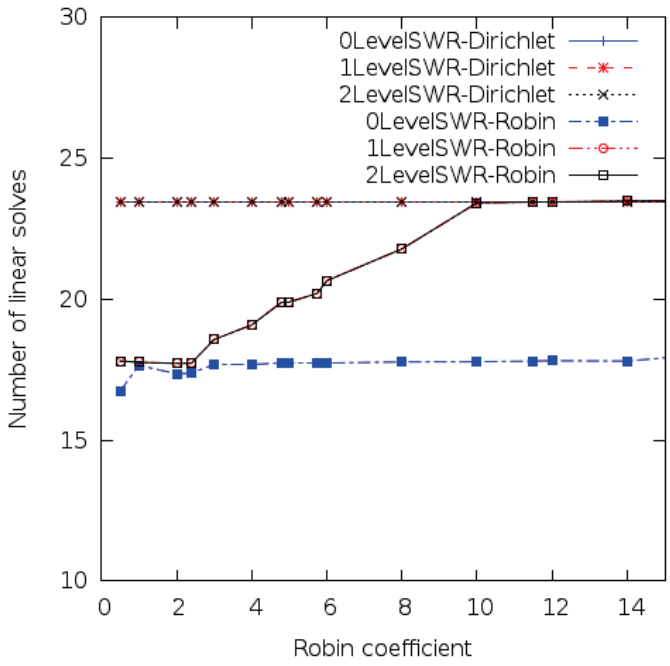


Figure 4.13: Average number of linear solves per iterations for 4 non-matching sub-domains. Zoom of fig 4.12.

4.2.3 Sound generation in a 2D low-Reynolds mixing layer

Another case presented here if the case of a 2D low-Reynolds mixing layer where a high precision scheme is required. It is studied especially focusing on the acoustic waves emitted by the vortex pairings in a perturbed mixing layer.

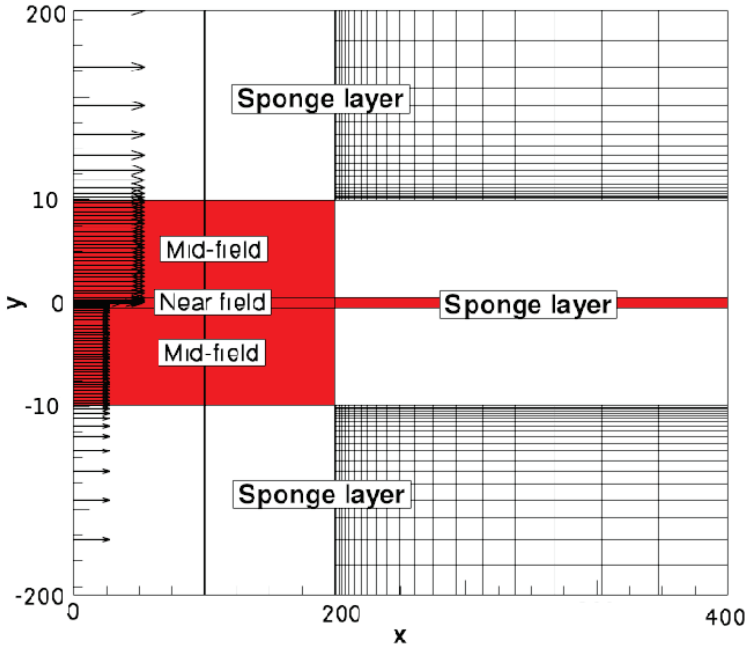


Figure 4.14: Mixing Layer acoustic pressure field. Computational domain with sponge layer.

The flow configuration is the same as the one proposed by Colonius and al [24] consisting

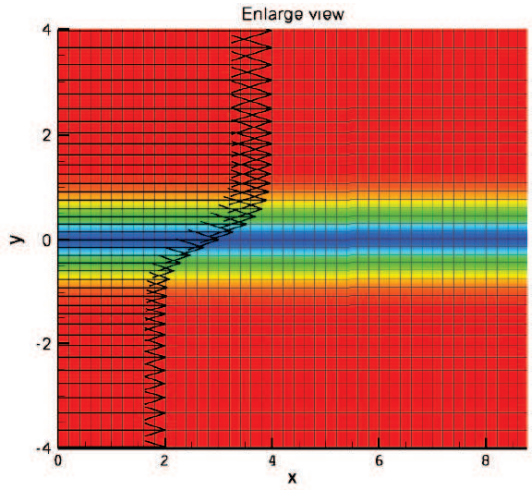


Figure 4.15: Mixing Layer acoustic pressure field. Initial domain (zoom of the fig.4.14).

in a slightly perturbed hyperbolic tangential shape velocity profile

$$u = \bar{u} + 0.125 \tanh(2y),$$

with $\bar{u} = (u_\infty + u_{-\infty})/2$ and $u_\infty = 0.5$, $u_{-\infty} = 0.25$, and $\rho_\infty = \rho_{-\infty} = 1$ and $p_\infty = p_{-\infty} = 1/\gamma$, respectively, with $\gamma = 1.4$. We fix the Reynolds number at 250 and add a sponge layer as shown in Fig.4.14 to absorb the flow. This is a particularly sensitive case in acoustics and phenomena are quite different within each sub-domain.

This is a particularly sensitive case in acoustics and it is an example of domains with regions of high activity and regions close to equilibrium for which the use of adaptive time steps is quite pertinent.

On fig.4.16, fig.4.17 and fig.4.19 we show the evolution of the solution found with the Navier–Stokes solver. We show the vorticity and the acoustic pressure. We observe the appearance of vortices, which ends rolling on themselves. In the last figure (fig.4.19) we can see the absorbing effect of the sponge layer. The diffusion is entirely numerical due to the coarse mesh inside the sponge region. On fig.4.17 we show the solution at instant $t = 200$ found with the Euler solver. We observe some kind of perturbation that seems to appear from the outflow. We have no explanation for this phenomena, but, it was observed by other researchers as well ([55]).

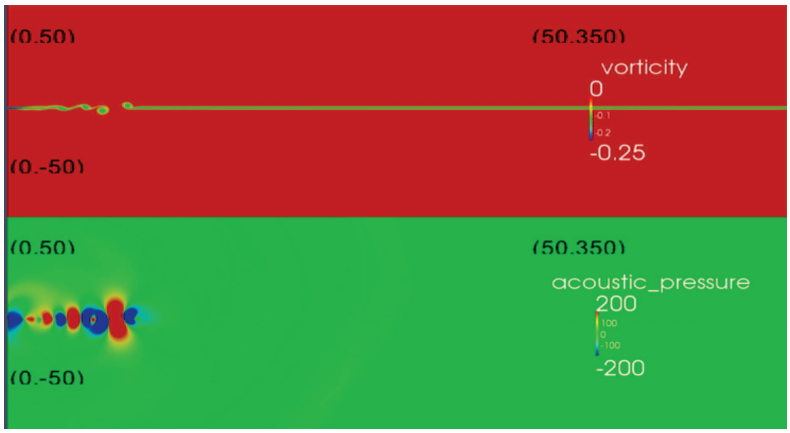


Figure 4.16: Mixing layer acoustic pressure field (top) and vorticity (bottom). Zoom over the computational domain. Solution of the Navier–Stokes equations with and 1 level adaptive SWR method. Plot at instant 100s.

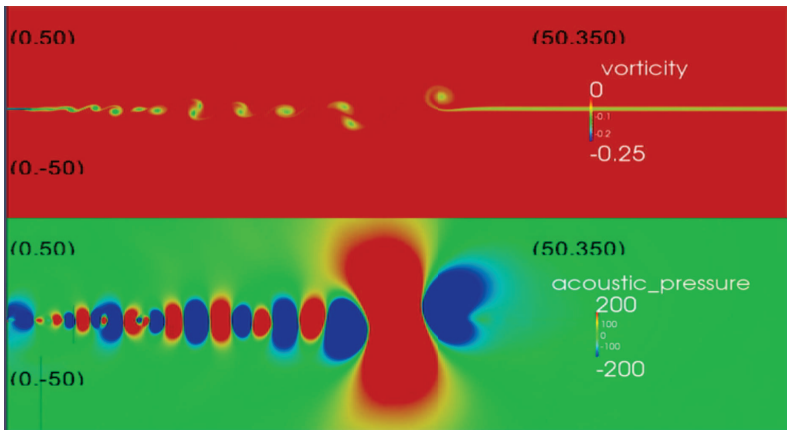


Figure 4.17: Mixing layer acoustic pressure field (top) and vorticity (bottom). Zoom over the computational domain. Solution of the Navier–Stokes equations with and 1 level adaptive SWR method. Plot at instant 200s.

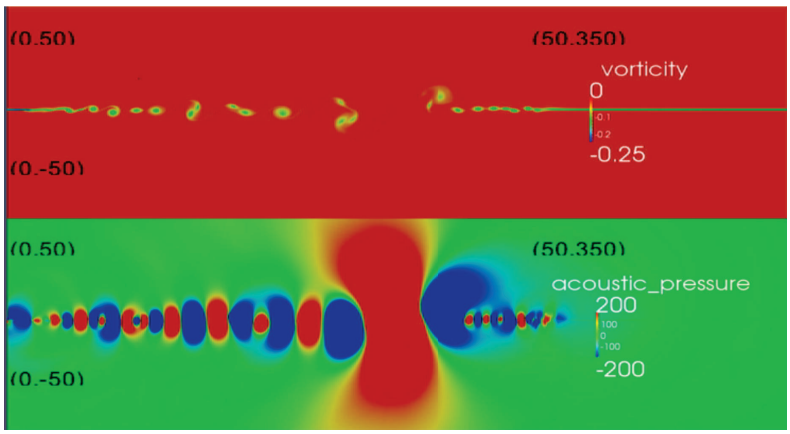


Figure 4.18: Mixing layer acoustic pressure field (top) and vorticity (bottom). Zoom over the computational domain. Solution of the Euler equations with and 1 level adaptive SWR method. Plot at instant 200s.

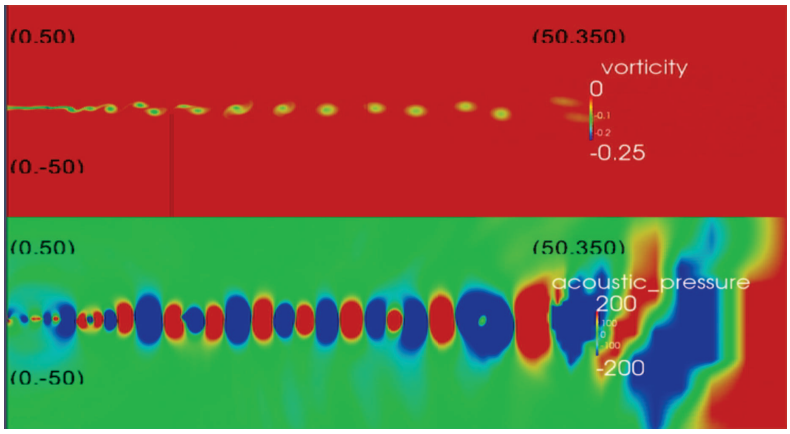


Figure 4.19: Mixing layer acoustic pressure field (top) and vorticity (bottom). Zoom over the computational domain. Solution of the Navier–Stokes equations with and 1 level adaptive SWR method. Plot at instant 400s.

Our results compare well with those obtained with an explicit third order Runge Kutta Discontinuous Galerkin solver developed by L. Halpern, J. Ryan and M. Borrel in [55].

At the beginning of the simulations, for a large number of iterations any perturbation crosses the sub-domains interfaces. The presented results are issued of simulations started at instant $t = 200s$ and ended at instant $t = 300s$. The perturbed solution was computed with an explicit second order Runge-Kutta method. We have fixed the stopping criterion in the Newton algorithm to a tolerance of $1.e-4$ and the stopping criterion of the Schwarz decomposition to a tolerance of $1.e-2$, what we believe it gives an acceptable approximated solution.

The global domain is first separated in four regions, the interest fine region of size $[0, 200] \times [-10, 10]$, a first sponge region $[200, 400] \times [-10, 10]$ and two extreme sponge regions of sizes $[-200, -10] \times [0, 400]$ and $[10, 200] \times [0, 400]$. The global domain was divided in 22 sub-domains: 18 sub-domains of equal size 107×21 cells in the middle region and 4 sponge sub-domains with 107×41 in the sponge area. In the sponge area, the space steps are computed following a geometric progression from left to right and from the middle of the global domain to the bottom and the top of it. Inside the fine sub-domains the space steps are of equal size. The sponge sub-domains are chosen to be two times greater since their time steps will be greater than the time steps in the fine region, the region of interest in our case.

The linear solves is no longer a good measure since sub-domains with different size have been computed. In 4.5 we vary the time window length and show only the total computational time cost for four methods: Newton-Schwarz, Schwarz-Newton, SWR with one level of adaptivity and the SWR with two levels of accuracy.

Table 4.5: Global computational costs for Schwarz tol = 1.e-2 and Newton tol=1.e-4

	$\Delta T = 5\delta t$				
Scheme \cfl	0.5	1	2	5	10
Newton-Schwarz	333.56	167.65	116.45	24.20	18.79
Schwarz-Newton	129.97	76.07	89.41	26.76	10.45
1 Level-SWR	189.13	189.65	121.77	21.90	5.87
2 Level-SWR	189.82	191.08	121.54	21.86	5.12

SWR appears more efficient as soon as we increase either the cfl or the time window size.

On fig.4.6 we compare results found with overlap $\theta = 3$ cells for a cfl stability criteria equal to 1. We used the same four regions of computational domain (of global size 600×550) and divide the domain in 14 sub-domains. In order to be coherent, the computation starts at time instant $t = 50s$. Results are first found with a Dirichlet boundary condition and second with a Robin boundary condition. The Robin coefficient was chosen equal to $\frac{1}{\sqrt{min(\Delta x, \Delta y)}} = 0.82$.

Table 4.6: Global computational costs for Schwarz tol = 1.e-2 and Newton tol=1.e-5 and for 1s of physical time.

	$\Delta T = 5\delta t$	
Scheme \Boundary	Dirichlet	Robin
Newton	467,21	474,72
Newton-Schwarz	1469,53	1448,45
Schwarz-Newton	454,82	501,03
0 Level-SWR	579,18	581,24
1 Level-SWR	374,17	369,35
2 Level-SWR	377,40	370,12

For this configuration, the computational cost found with the Robin boundary condition are comparable with the ones found with the Dirichlet boundary condition.

4.3 Vortex shedding from rectangles

This last test has the purpose of presenting numerical solutions for stiffer problems, proving that the proposed schemes adapt well to more complicated configurations and work for high number of sub-domains. Computations have been carried out for 2D time-depending laminar flow around rectangles in infinite domains. The unsteady character of the flow and the vortex shedding motion are making the numerical calculations very difficult. The numerical stability condition must be very low (cfl=.2 for explicit methods and cfl=1 for implicit methods in our computations).

We consider that the flow starts from rest. As vorticity diffuses out from the body surface the boundary layer thickness until the flow separates. This phenomena appears almost instantaneously at the corners of the obstacle. Next, as the vorticity keeps diffusing, a pair of vortices are formed and curl up on themselves. Next, an asymmetry develops and vortices of opposite signs are shed alternatively. Von Karman was the first to analyse and interpret the vortex street as an intrinsic property of wake structure.

The initial solution and inflow conditions are uniform, non-fluctuating velocity profile. Transmissible boundary conditions are imposed in the bottom, top and the right boundaries of the computational domain. Around the rectangle no-slip conditions are imposed. This condition is ensured by a refined grid around the rectangular body.

Studies of flow past rectangular prisms have been made for a wide range of real life applications [28, 9, 80, 92, 86] like the aerodynamic drag reduction for air-planes, road vehicle, damage predictions for inclined air-foils, ocean pipe line or risers, off-shore platform supports, suspension bridges, steel towers or smoke stacks, etc. The need for this kind of simulations in the case of aerodynamic drag reduction for air-planes is to ensure suitable periods of elapse between air-crafts landings. Tests with moving rectangular micro-prism has been conducted with the aim of energy harvesting microresonator based on the forces generated by the Karman street around the bodies. In turbulent regime, due to the large frequency range of turbulence for high Reynolds numbers, accurate resolutions are difficult to obtain with large eddy simulations, ideally a direct numerical simulation should be used.

Different behaviours can be found for high or low Reynolds numbers, different size of bodies, angle of attack.

Initial configuration. Let us fix the domain size at $\Omega = [0., 30.] \times [-6., 6.]$ as on fig.4.20. We add an unit rectangular solid of origin $(4., -0.5)$, centred in the vertical direction of the domain. We fix a uniform initial solution of primitive variables (ρ_0, u_0, v_0, p_0).

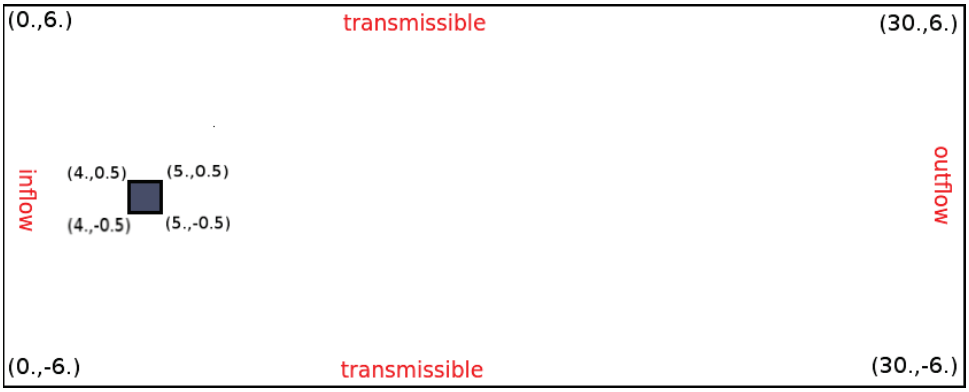


Figure 4.20: Computational domain of size $\Omega = (0., 30.) \times (-6., 6.)$. Centred in the vertical direction we place a rectangle of unit size and coordinates $\Omega_{solide} = (4., 5.) \times (-.5, .5)$.

$$\begin{cases} \rho_0 & = 1. \text{ the initial density,} \\ u_0 & = 1. \text{ the initial velocity in the x-direction,} \\ v_0 & = 0. \text{ the initial velocity in the y-direction,} \\ Ma & = 0.33 \text{ the initial Mach number,} \\ \gamma & = 1.4 \text{ the ratio of specific heat for an ideal gas,} \\ p_0 & = \frac{\rho u^2}{\gamma Ma^2} \text{ the initial pressure,} \\ Re & = 250 \text{ the Reynolds number.} \end{cases}$$

At each time step we impose an inflow of equal values to the initial solution. We suppose that the domain is infinite in the x direction. At right, top and bottom we impose transmissible boundaries conditions. At the interfaces around the rectangle we impose no-slip conditions: $\vec{u}.n = 0$ and $\nabla\vec{u} = 0$.

For sharp-edged bodies numerical problems may appear at the corners, mathematical singularities yield physically inaccurate flow properties. From a numerical point of view these singularities should be treated separately with a more precise numerical scheme than in the rest of the domain. In our case we minimise as possible this phenomena using very fine meshes around the body (represented on fig.4.21). Another sensibility is coming from the structured meshes that force us to separate sub-domains in the corner of the rectangle and impose oscillations at artificial boundaries, another difficulty that the transmissible boundary condition must solve. This problem could be solved for unstructured meshes.

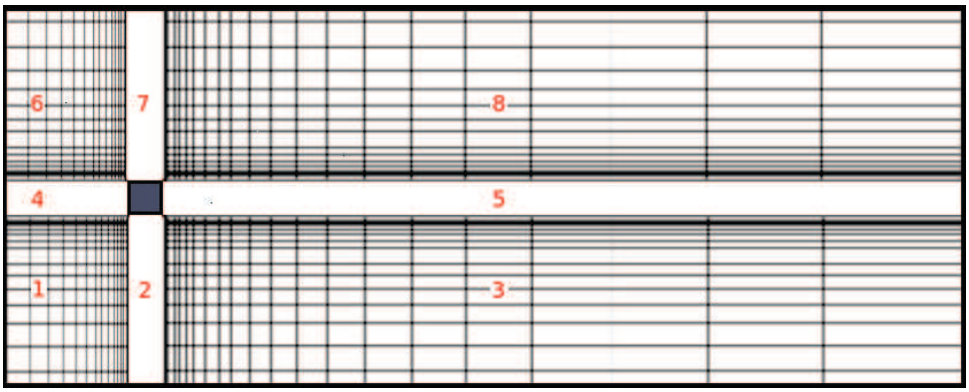


Figure 4.21: Computational domain mesh. The mesh is more refined around the body because of large flow oscillations in this region. We distinguish 8 different regions, each one of them can become a sub-domain or be further partitioned in sub-domains.

Figures 4.22 to 4.25 shows the flow evolution in time from 2.94 seconds to 50 seconds. The solution of the Navier–Stokes system of equations was built with a second order Runge Kutta method. The time step was recomputed after each time update to satisfy a CFL number equal to 0.2. We note that, for a CFL number greater the 0.2 the solution failed to converge due to numerical instabilities, implicit methods are required.

On figure 4.22, we observe a separation of two opposite sign vorticity flows. Two vortices are forming, but they do not seem to interact. Although the body is vertically centred, after few more iterations one vortex (fig.4.23) becomes more important and dominates the other. The two vortices separate and curl up on themselves. The phenomena is continuously reproduced (fig.4.24 and fig.4.25) until we observe a stabilised phenomena where opposite vortices are rolling one after another. We observe regions of concentrated vorticity. Moreover, the pressure variation along the side walls of the body is much higher than the one along the bottom base and top walls.

According to the initial mesh splitting, the computational domain is partitioned in 20 sub-domains as following:

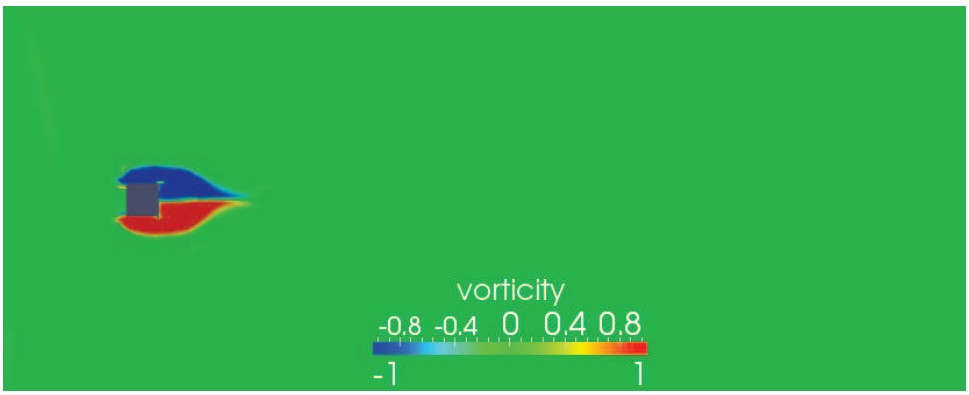


Figure 4.22: Vorticity flow past a rectangle. Intermediary solution plotted after 1000 iterations, equivalent to the instant $T = 2.94s$ of the simulation. Around the square, two opposite vorticity sign flows are separated.

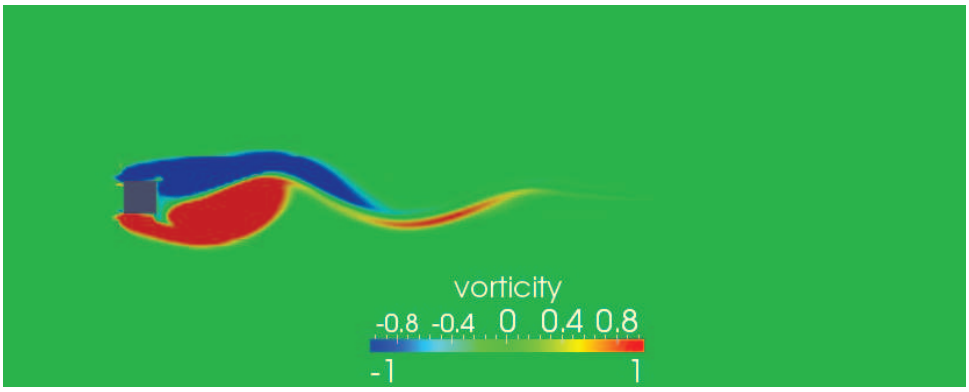


Figure 4.23: Vorticity flow past a rectangle. Intermediary solution at instant $T = 14.20s$ computed with an explicit second order Runge Kutta scheme. The bottom vortex increases faster then the above one and starts rolling over this last one.

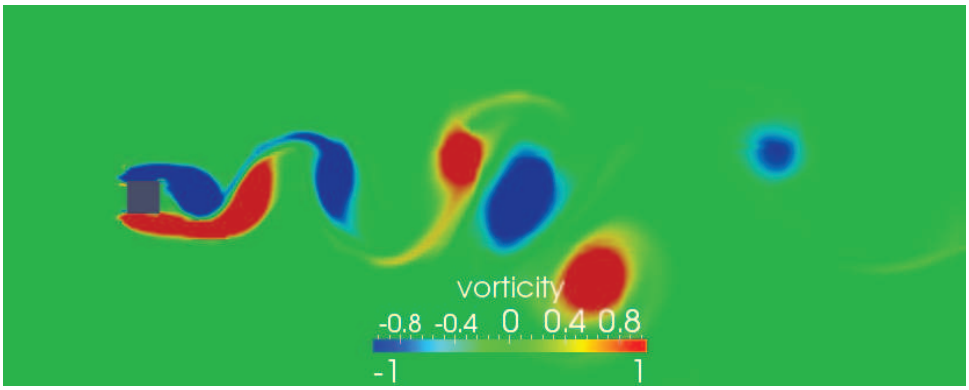


Figure 4.24: Vorticity flow past a rectangle. Intermediary solution at instant $T = 29.41s$ computed with an explicit second order Runge Kutta scheme. We observe four completely separated vortices, a fifth one that is almost separated and two others that begin to curl on themselves.

- Region 1 becomes $\Omega_1 = [0., 4.] \times [-6., -0.5]$ of size 80×120 cells;
- Region 2 becomes $\Omega_2 = [4., 5.] \times [-6., -0.5]$ of size 40×120 cells;
- Region 3 is split according to the x direction into 5 sub-domains $\underset{3 \leq i \leq 7}{\cup} \Omega_i = [5., 30.] \times$

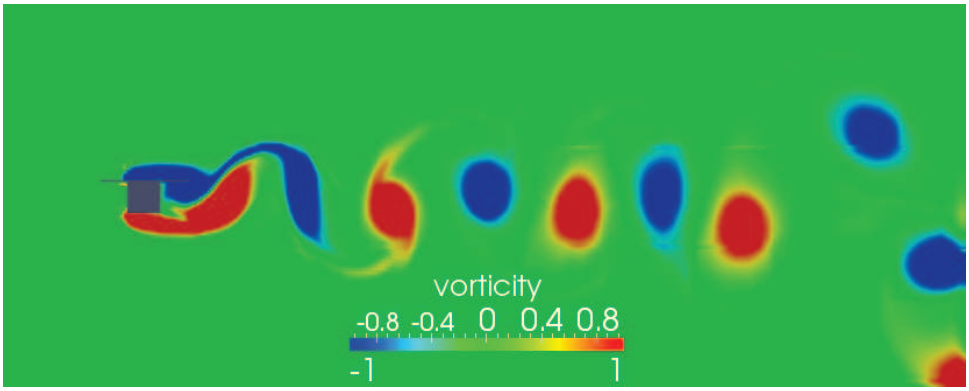


Figure 4.25: Vorticity flow past a rectangle. Solution at final time $T = 50s$ computed with an explicit second order Runge Kutta scheme. Some vortices are leaving the domain, others are stabilised and roll one after each other.

$[-6., -0.5]$ of total size 500×110 cells;

- Region 4 becomes $\Omega_8 = [0., 4.] \times [-0.5, 0.5]$ of size 80×40 cells;
- Region 5 is split according to the x direction into 5 sub-domains $\cup^{9 \leq i \leq 13} \Omega_i = [5., 30.] \times [-0.5, 0.5]$ of total size 500×30 cells;
- Region 6 becomes $\Omega_{14} = [0., 4.] \times [0.5, 6.]$ of size 80×120 cells;
- Region 7 becomes $\Omega_{15} = [4., 5.] \times [0.5, 6.]$ of size 40×120 cells;
- Region 8 is split according to the x direction into 5 sub-domains $\underset{16 \leq i \leq 20}{\cup} \Omega_i = [5., 30.] \times [0.5, 6.]$ of total size 500×110 cells.

We get a total of 149900 mesh cells, without counting the additional overlap and fictitious overlap (ghost cells). Although, the sub-domain sizes are no longer equal, we decide to take a look at the average number of linear solves for simulations with Dirichlet and Robin type transmission conditions. We are interested in a general view over the computational cost and more particular we are searching for the more expensive sub-domain. We discuss solutions of the Navier–Stokes equations found with the second order implicit BDF. The convective fluxes are computed with a second order MUSCL method combined with an AUSM$^+$−up scheme and a Van Albada limiter. The linear system is solved with the relaxation method in 2 iterations. For each Schwarz process the relative tolerance is of $1e-3$. We ask the Inexact Newton process to stop when tolerance of $1e-5$. Based on previous discussions we consider that, for inexact methods, there is no need to ask for a smaller tolerance.

For the classical parallel methods, Newton (or Newton-Schwarz with one Schwarz iteration), Newton-Schwarz and Schwarz-Newton we plot the minimum (fig.4.28), maximum (fig.4.27) and overall average number (fig.4.26) of linear solvers needed to compute one iteration. When the partitioned Newton or Schwarz-Newton method is used, we get the expected result, the most active sub-domains are those close to the bluff body and the less expensive are those far away from the body. We have manually unbalanced the sub-domains to try to synchronise their tasks. In what concerns the resolution with Robin transmission conditions, we can say only that for Robin coefficients with values between $\frac{1}{\sqrt{\min(\Delta x, \Delta y)}} = 3.4$ and $\frac{1}{\min(\Delta x, \Delta y)} = 12.14$ is slightly more efficient or as efficient as the resolution with the Dirichlet type boundary condition. For Schwarz-Newton method, the use of a Robin type transmission condition is less efficient for all tested coefficients. Moreover, all sub-domains require the same number of linear solvers. In this case the computational cost equals the computational cost of one of larger size sub-domain ($\Omega_i, 3 \leq i \leq 7$ and $16 \leq i \leq 20$) and the communication cost between sub-domains. One would get a better result for well-balanced sub-domains. I believe that these better results for

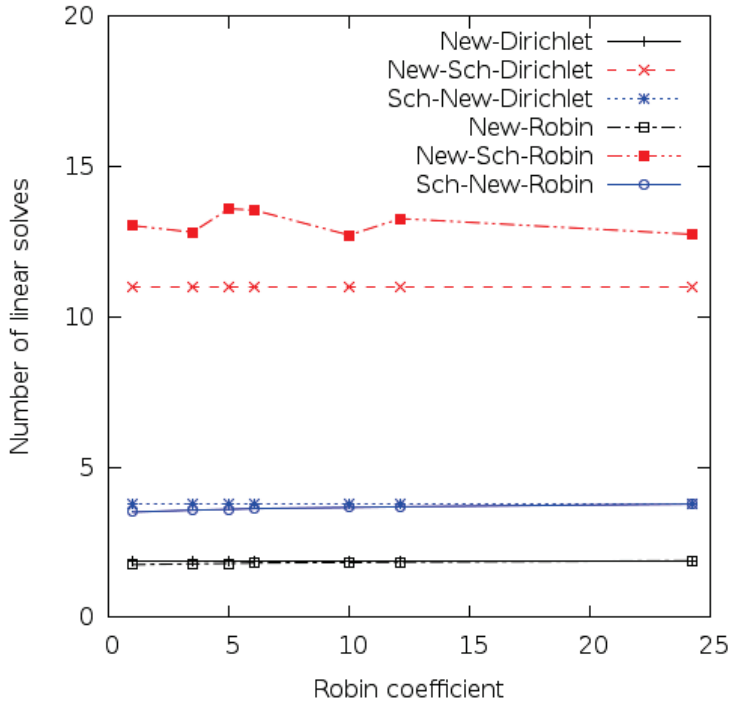


Figure 4.26: Average number of linear solves per time iteration and over all sub-domains.

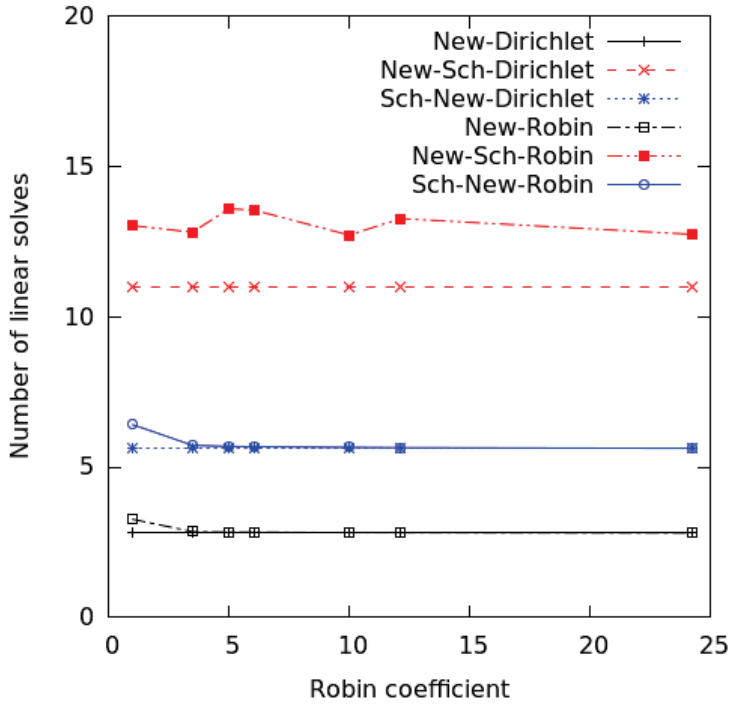


Figure 4.27: Average number of linear solves per time iteration for the sub-domain 8 (right of the rectangle). This sub-domain is the one that needs the biggest number of linear solves to achieve the desired relative error. For the Newton-Schwarz method all sub-domains need the same number of linear solves.

well-balanced sub-domains is due to the global character of the non-linear solver which is more sensitive than the linear solver.

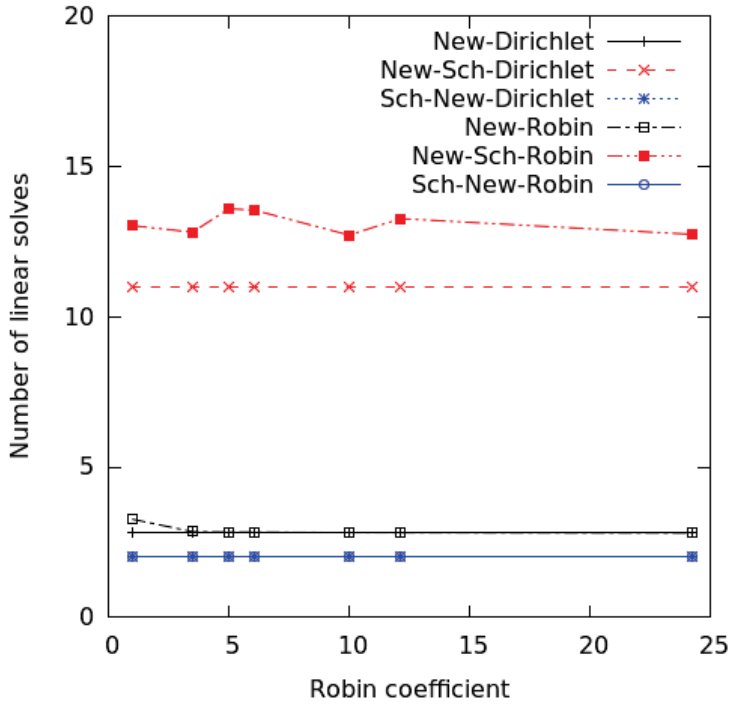


Figure 4.28: Average number of linear solves per time iteration for the sub-domains 6, 12 and 19. Strangely, the less number of linear solvers is required not by the last columns of sub-domains, as was expected, but the column before the last one. For the Newton-Schwarz method all sub-domains need the same number of linear solves.

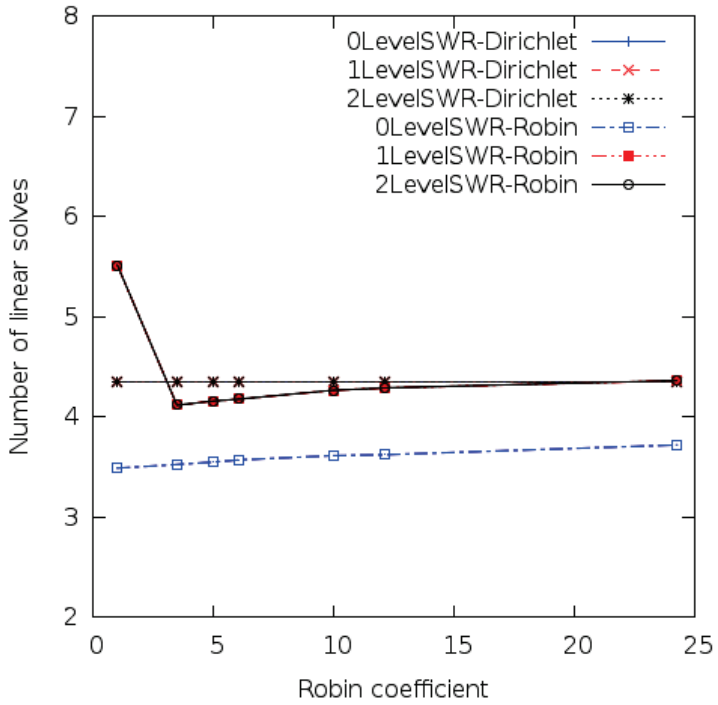


Figure 4.29: Average number of linear solves per time iteration and over all sub-domains.

The SWR methods were computed with the fixed time window of size 5 times the smallest global time step, $\Delta T = 5\delta t$. When Dirichlet type transmission conditions are used, there is no

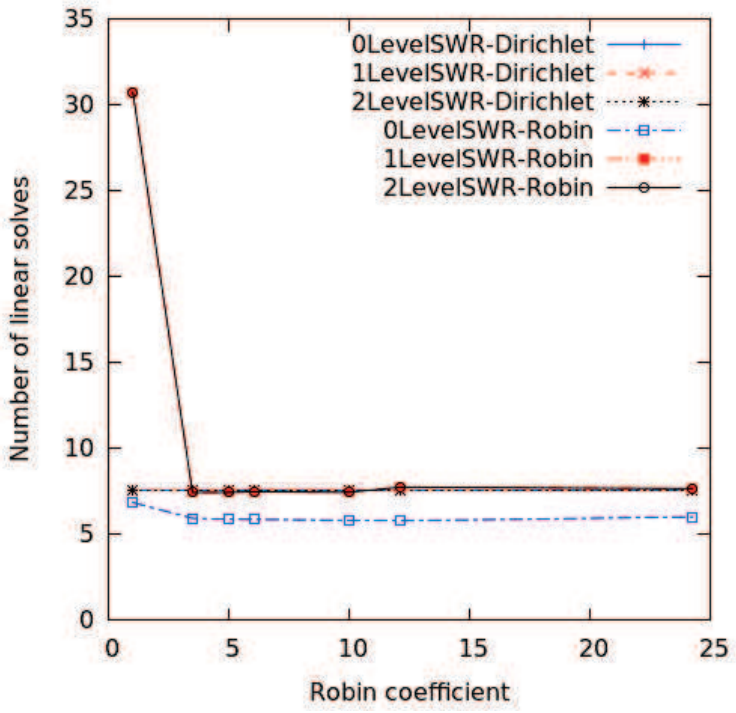


Figure 4.30: Average number of linear solves per time iteration for the sub-domain 8 (right of the rectangle). As for the classical methods, the sub-domain 8 is the one that requires more computations.

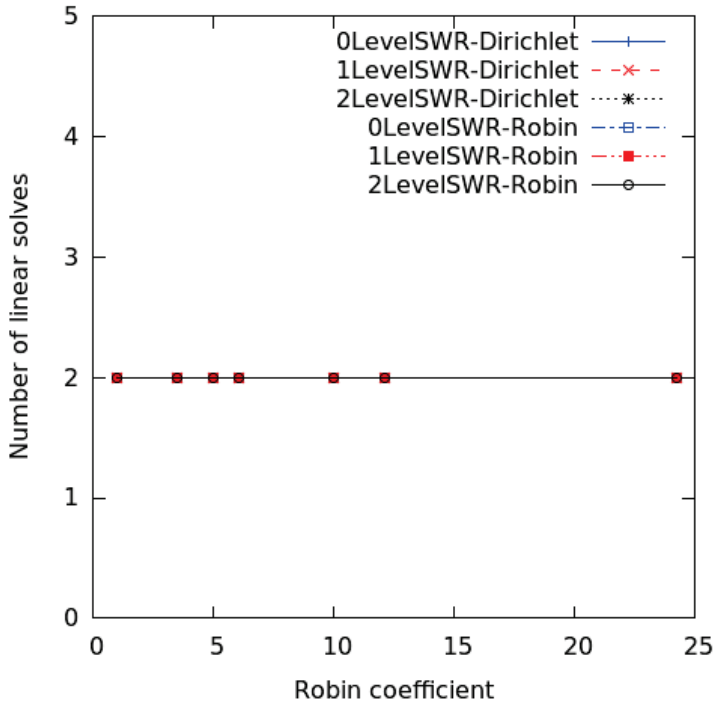


Figure 4.31: Average number of linear solves per time iteration for the sub-domains 6,12,19. These sub-domains are the one requiring the smallest number of linear solvers at it is the same for resolutions with Dirichlet or Robin type boundary transmissions.

difference between the two adaptive SWR methods and the one with fixed time disretisation.

The SWR method with the same fix time step in each sub-domains gives the best results in terms of number of linear solvers. The Robin transmission condition reduces the costs for the same interval range $\left[\frac{1}{\sqrt{\min(\Delta x,\Delta y)}}, \frac{1}{\min(\Delta x,\Delta y)}\right]$ of the Robin coefficient The Robin coefficient λ_{Robin} from $\frac{\partial u}{\partial n} + \lambda_{Robin} u = g$ is globally computed.

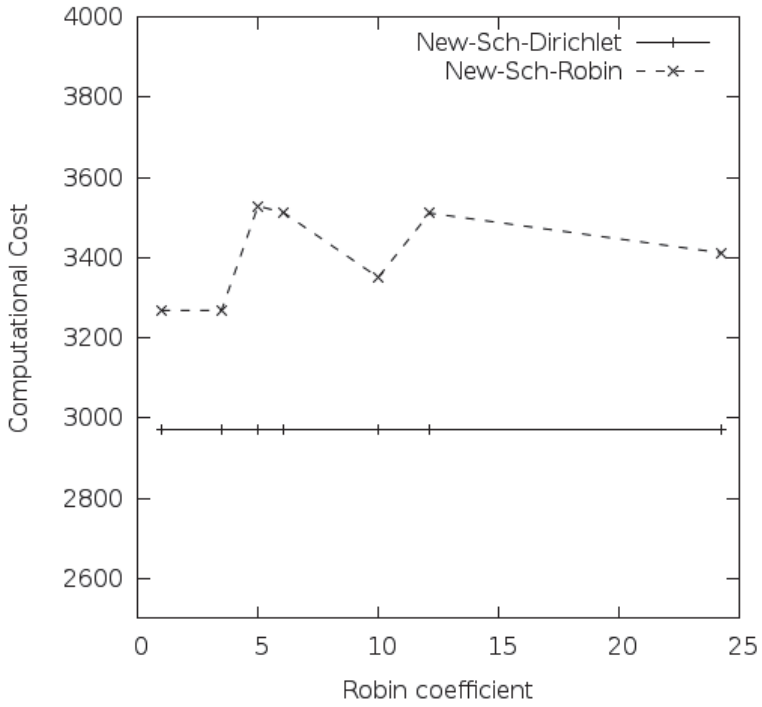


Figure 4.32: Computational cost (in seconds) to achieve one second of flow evolution with Newton-Schwarz scheme. We give results for different values of Robin coefficient.

Although, the average number of linear solves per iteration does not oscillate very much with different Robin coefficient, when we take a look at the computational cost of one second (fig.4.32, fig.4.33 and fig.4.34) we see that it can have a big impact. After having seen that the Newton-Schwarz method has the highest number of linear solves we expect it to be the most costly. Indeed, it is the result that we obtain on fig.4.32, it needs almost one hour to reach one second. And this results get worse when Robin conditions are used, independently of the Robin coefficient. Eventhough, the Robin condition reduces only slightly the number of linear solver per iteration, when all these reductions are cumulated we obtain an important reduction of the computational cost. The difference between the computational cost of a partitioned Newton or Schwarz-Newton methods resolution with Dirichlet transmission conditions, and the same methods with Robin transmission condition and best Robin coefficient increases in time.

In the 1 level and 2 levels of adaptivity, the SWR method is very expensive for a small Robin coefficient (0.5 on the figure). They have the best cost reduction for Robin coefficients in the interval $[\frac{1}{\sqrt{\min(\Delta x,\Delta y)}}, \frac{1}{\min(\Delta x,\Delta y)}]$. For a well chosen Robin coefficient the SWR method is the cheapest one. The results of the SWR with 0 level of adaptivity, meaning with fixed discretisation of each time window, and the same discretisation inside each sub-domain are comparable with those obtained by a Schwarz- Newton method with an optimal Robin condition. What happened? We have seen that the average number of linear solvers per iteration is smaller than for the adaptive SWR methods. But, the local time step is the same in each sub-domain, thus, each sub-domain computes 5 time steps (in this particular case) to reach the end of one time window. In the case of 1 level adaptive SWR, the smallest sub-domain can compute 5 or maybe even less time steps to reach the end of a time window, the largest sub-domain can also

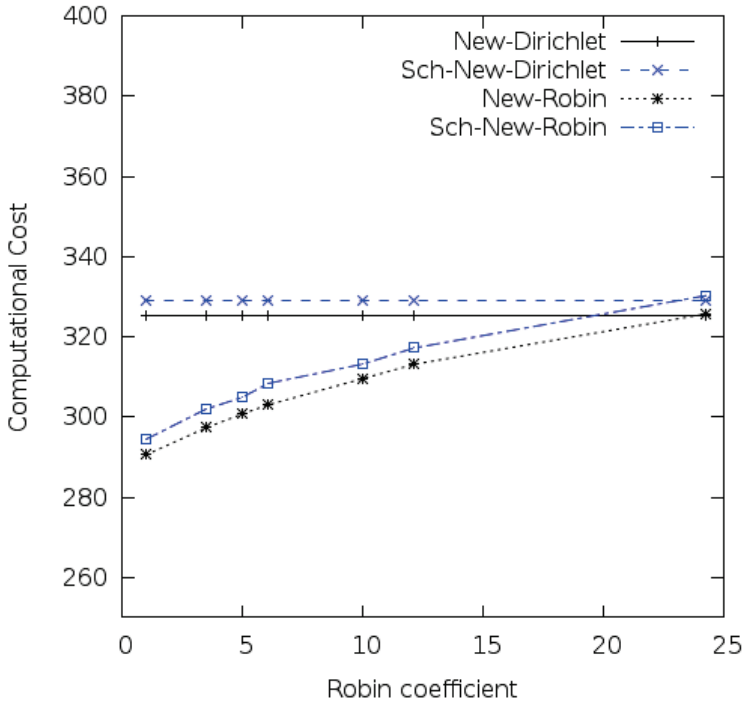


Figure 4.33: Computational cost (in seconds) to achieve one second of flow evolution with the partitioning Newton scheme and the Schwarz-Newton scheme. We vary the Robin coefficient.

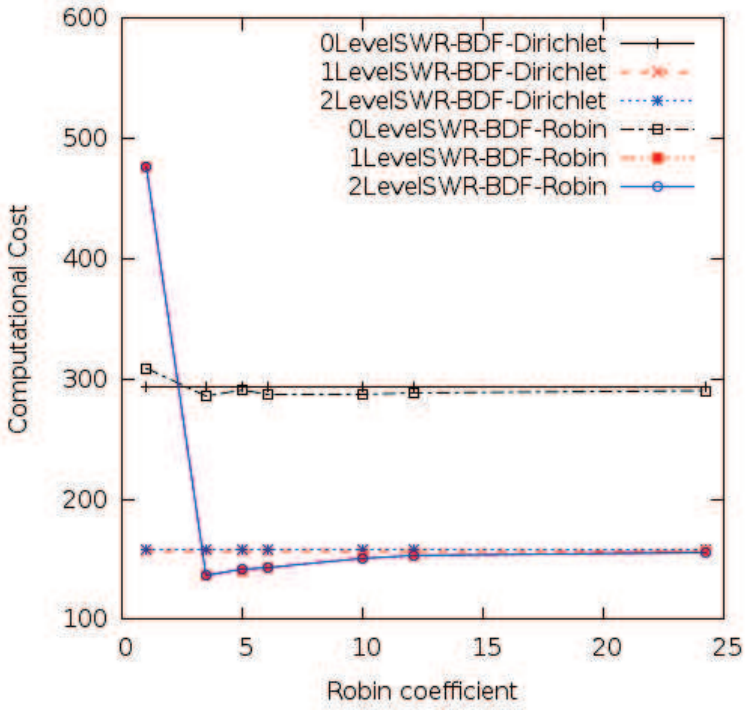


Figure 4.34: Computational cost (in seconds) to achieve one second of flow evolution with different values of Robin coefficient. Simulations are computed with all three, fixed and adaptive, SWR methods.

compute less than 5 time steps. And this is what happens when the CFL condition is locally satisfied, the far away sub-domains from the solid body reach a time window in 2 to 3 time

steps, instead of 5.

Nevertheless, the SWR methods, using any of the proposed boundary conditions reduce the computational costs to half the costs of a classical Schwarz-Newton method and much more over the of Newton-Schwarz method. This result is certainly due to the fewer number of communications between the sub-domains, but also to the local adaptivity of the time step. For this application, there is no difference between the 1 level and 2 level adaptive SWR methods.

The last simulation was conducted with the 104 sub-domains. The computational domain was partitioned as:

- Region 1 is split according to the $y-$direction into 3 sub-domains $\underset{1\leq i\leq 3}{\cup}\Omega_i = [0.,2.]\times[-6.,-0.5]$ of total size 100×150 cells;
- Region 2 is split according to the $y-$direction into 3 sub-domains $\underset{4\leq i\leq 6}{\cup}\Omega_i = [2.,3.]\times[-6.,-0.5]$ of tocal size 60×150 cells;
- Region 3 is split into 13×3 sub-domains $\underset{6\leq i\leq 44}{\cup}\Omega_i = [3.,30.]\times[-6.,-0.5]$ of total size 814×150 cells;
- Region 4 becomes $\Omega_4 5 = [0.,2.]\times[-0.5,0.5]$ of size 100×60 cells;
- Region 5 is split according to the $x-$direction into 13 sub-domains $4\underset{6\leq i\leq 59}{\cup}\Omega_i = [3.,30.]\times[-0.5,0.5]$ of total size 814×50 cells;
- Region 6 is split according to the $y-$direction into 3 sub-domains $\underset{60\leq i\leq 62}{\cup} = [0.,2.]\times[0.5,6.]$ of size 100×150 cells;
- Region 7 is split according to the $y-$direction into 3 sub-domains $\underset{63\leq i\leq 65}{\cup} = [2.,3.]\times[0.5,6.]$ of size 60×150 cells;
- Region 8 is split into 13×3 sub-domains $\underset{66\leq i\leq 104}{\cup}\Omega_i = [5.,30.]\times[0.5,6.]$ of total size 814×150 cells.

The computational domain contains 347040 cell, without counting the fictitious ones. Since a unit square is imposed closer to the inflow boundary, the vortices appear earlier. They can be seen on fig.4.35 and fig.4.36. On table 4.7, we only present results issued from Dirichlet type boundary condition. The time domain is fixed to $[0,10]$. The time window inside the SWR methods equals 5 times the smallest local time step. The results are consistent with the previous shedding test. The SWR methods reduce consistently the computational cost. The results from the two adaptive SWR methods are comparable and their computational cost is half the cost of the initial partitioning scheme. The classical SWR method, with fixed time steps is rather close to the Schwarz-Newton method in term of computational costs. The Newton -Schwarz method, hardly, manages to give a result and it is very expensive. This test allows us to validate the methods on large number of sub-domains, and show that the adaptive SWR methods are efficient on large number of sub-domains of rather small sizes.

Table 4.7: Global computational costs for Schwarz tol = 1.e-2 and Newton tol=1.e-5 and for 10s of physical time.

$\Delta T = 5\delta t$	
Scheme \Boundary	Dirichlet
Newton	14055.04
Newton-Schwarz	78894.81
Schwarz-Newton	14246.54
0 Level-SWR	7994.34
1 Level-SWR	8004.53
2 Level-SWR	13040.13

Figure 4.35: Vorticity flow past a rectangle. Solution at instant $T = 10s$ computed with the 2 level SWR implicit scheme. The unit square is closer to the border, at position $[2, -.5]$.

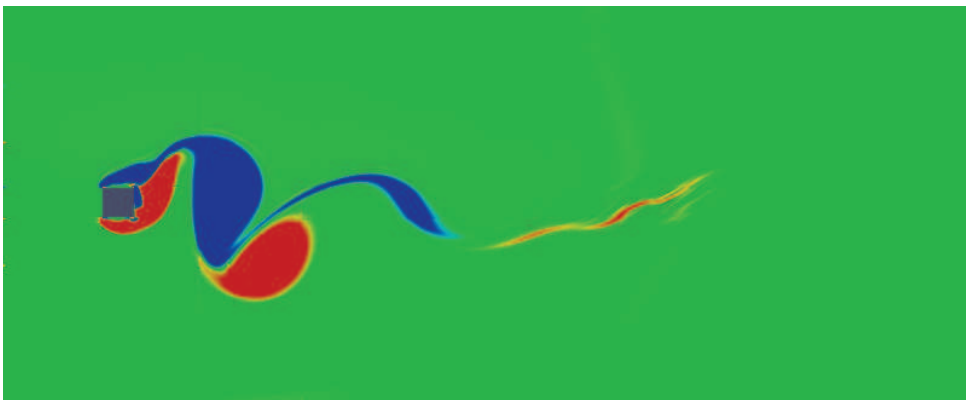


Figure 4.36: Vorticity flow past a rectangle. Solution at instant $T = 20.5s$ computed with the 2 level SWR implicit scheme. The unit square is closer to the border, at position $[2, -.5]$.

On table 4.8, we compare computational costs for simulations with Dirichlet and Robin type transmission conditions. The simulation begins at instant $t = 20$ seconds and ends after 0.5 seconds.

Table 4.8: Global computational costs for Schwarz tol = 1.e-2 and Newton tol=1.e-5 and for .5 seconds.

$\Delta T = 5\delta t$			
Scheme \Boundary	Dirichlet	Robin (coeff= $\frac{1}{\sqrt{\min(\Delta x,\Delta y)}}$)	Robin (coeff= $\frac{1}{\min(\Delta x,\Delta y)}$)
Newton	960.42	948.58	954.82
Newton-Schwarz	3184.07	14400.83	10320.02
Schwarz-Newton	982.15	981.82	984.57
0 Level-SWR	924.28	921.96	920.96
1 Level-SWR	562.59	576.24	528.18
2 Level-SWR	562.46	576.92	528.96

The last result presented in this work, on table 4.9, is found with the explicit Navier–Stokes solver. We have compared the computational cost with the second order Runge Kutta method and the SWR methods applied to the second order Runge Kutta method. The tolerance in the Schwarz stopping criteria equals $1.e-2$. The computation begins at $t = 20s$ and stops at $t = 20.5s$. The transmission conditions are of Dirichlet type. The adaptive SWR methods reduce almost by half the computational cost. The diminishing ratio is similar with the one

found with the BDF method.

Table 4.9: Global computational costs with the second order explicit solvers for a duration of 0.5 seconds.

$\Delta T = 5\delta t$	
RK2	3382.06
0 Level RK2	3168.98
1 Level RK2	1830.68
2 Level RK2	1832.12

To conclude in this chapter, the uniform flow simulations were used as applications to check that the code is fully operational, to check the efficiency of classical methods and the one of the improved domain decomposition methods. We were mostly interested in the analysis of the impact of the SWR methods over the total computational cost. We observed that, in simulations that apply to the SWR methods, the adaptive SWR methods consistently reduce the computational cost. We mean simulations where the computational domain is split in sub-domains with different level of activity, such as the mixing layer case or the vortex shedding around rectangles. The case of the 2D isentropic vortex allowed us to study the accuracy and the convergence of the SWR methods.

Conclusion and Perspectives

To reproduce and understand real-life fluid dynamics simulations, issued of phenomena more and more complex, requires the treatment of an increased amount of data and higher numerical precision. Although we dispose of a large variety of computer resources, architectures and capabilities, the main issue remains the memory limit and huge computational costs. Numerical algorithms that replicate phenomena with high fidelity are no longer sufficient. They must be conceived to use and adapt to all existent hardware. During this three years period we were interested in the parallel approach of a direct numerical simulation of compressible flows. A CFD code was built to be easily incorporated in any industrial code, starting with the Onera's software elsA. We have studied the resolution of non-linear problems resulting from explicit and implicit methods that discretise the Euler and the compressible Navier–Stokes equations. We restrained ourselves to the study of the finite volume method applied to Cartesian meshes, often used in CFD codes.

Everything should be made as simple as possible, but no simpler (Einstein). Our aim was an easy and fast incorporation in any industrial code. It means that classical solvers must remain as intact as possible. We needed to find methods that efficiently parallel compute a Navier–Stokes solver with as little modification as possible. Inside elsA, the parallelisation is made using partitioning methods, overlapping or non-overlapping sub-domains and communicate by messages that are stored in fictitious mesh cells. It allows each proper cell (except fictitious cells) to the domain to be computed with the same accuracy order as any interior cell. This was our main strategy: use fictitious or ghost cells to fulfill physical and artificial boundary conditions. Fictitious cells imply dealing with overlapping sub-domains. The Schwarz based domain decomposition methods are well suited to parallel computing overlapping domains. They were studied for both explicit and implicit numerical schemes in the second chapter of this manuscript. When implicit schemes are used to solve the Navier–Stokes system of equations (the second order BDF method), the algorithm can be seen as a three step algorithm: i) discretisation in time, ii) solving the resulting non-linear system and iii) solving the resulting linear system. Three different methods were immediately distinguished: partitioning Newton, Newton-Schwarz and Schwarz-Newton. As their name suggest, they act on steps ii) and iii) of the algorithm. The first one is used in elsA, the next two are popular algorithms, especially when Krylov methods are used to solve the linear system of equations. A fourth method, discussed in chapter II, is the SWR method and adds the time scale into the parallelisation. It decomposes the global domain even before the beginning of the algorithm. First, the domain decomposition is made and second, each sub-domain is solved in parallel with its local time discretisation, non-linear and linear solver. Time windowing techniques were created to ensure stability over the global domain. Our attention focused on this method and two more flexibilities were proposed. The first one, called 1 level SWR, automatically computes local time steps at the first window iterate. The second one, called 2 level SWR, automatically adapts local time steps inside each window iterate.

Because of its flexibility, the SWR methods adapt to different computer resources. On distributed memory (MPI) they are optimal, as sub-domains, located on different processors, do not communicate at each time step, but only at the end of a time window. Sub-domains

can be of different size and the SWR methods are still efficient. Locally, parallel techniques (loop parallelisation with OpenMP) can be done on a maximum number of available processes. In chapter III, we discussed the implementation of these methods on shared, distributed and mixed memory. The GPU was the most recent notion of architecture. These many architectural levels force the developer to review its algorithm before deciding on its portability. We have implemented and illustrated a simple mono-domain second order Runge Kutta solver, working on the Nvidia GPUs (CUDA programming). We were able to decide which one of the domain decomposition method is most adapted to GPU programming. The communications between CPU and GPU are the most expensive step of a multi-GPU implementation. The SWR methods are built to minimise communications between sub-domains, thus they become very interesting on multi-GPU.

Moreover, convergence can be improved in a simple way, by increasing the overlap, or, with the use of higher order transmission conditions, such as Robin type transmission conditions. The classical treatment of a Robin type boundary condition is with the weak variational formulation of the problem. We decided to remain coherent with elsA's discretisation strategy and, instead of changing the integral formulation of the problem, we impose transmission conditions, resulting from Robin conditions, inside fictitious cells.

In the last chapter, we presented results and validation of numerical Euler and Navier–Stokes schemes and Schwarz based domain decomposition methods through systematic comparisons of existing schemes. We simulated 2D multi-scale Euler and Navier-Stokes problems. The results show that the SWR methods have the ability to treat large data systems without loss of parallel efficiency. The SWR algorithms have proved themselves more efficient than the Newton-Schwarz scheme. When the adaptivity is inefficient, the adaptive SWR methods are at least as efficient as the classical SWR method. The efficiency of the classical SWR methods is at least comparable to the one of the Schwarz-Newton method. If the application implies sub-domains of different size and discretisations, the adaptive SWR methods are cheaper and more efficient. The adaptive SWR with one level of adaptivity seems as efficient than the one with two levels of adaptivity. They have similar computational efficiency and adds a new flexibility to the SWR method.

The most important and novel contributions are: the adaptation of the SWR methods to the Cartesian Euler/Navier–Stokes solver, and the new flexibilities of the SWR methods.

We remark that, when dealing with large cases, industry is more likely to retain simple and slower techniques than complex and faster. Thus, research should focus more on automating methods in order to become interesting for industrial applications.

Perspectives

We would like to acknowledge to the reader that much improvement is still possible and that, through the spectrum of our acquired experience in this field and on recently published work we identify and propose the next points.

The conclusion of this work can not be definitive, as there are at least three ways to improve the SWR techniques. One is to keep optimizing the time space interface conditions, for example by trying absorbing boundary conditions or other conditions that contain informations depending on the time scale. In the recent International Conference of Domain Decomposition Methods, in September 2013, Ong and co. (see [82]) presented another way to improve the classical SWR method, which is also valid for the adaptive SWR algorithms. They propose to implement each SWR iteration in parallel using the pipeline. In a few words, it means that inside a window computation, once the first iteration produces enough interface conditions a second iteration can begin before the end of the first one. Moreover, we have presented different ways of adapting the time step, and it is usually dependent on a cfl number both for explicit and implicit methods in order to ensure a numerical stability. This cfl number is fixed in the

beginning of our simulation, another way would be to readapt this cfl number after each window iteration. About the implementation, very little modifications are to be done to the existing code. The use of SWR methods offers the possibility of using different discretisations in each sub-domain (coupling structured with unstructured meshes) and even coupling Navier–Stokes solvers with Euler solvers.

Using a multi-grid method or coarse grid correction (equivalent to a two-level multi-grid method) could enhance the overall efficiency of the all overlapping domain decomposition methods.

Another next step of this study is to be completed with a 3D industrial code. This should be easily done as the initial structure of each module is already created to support a third dimension. A greater challenge code modification would be the implementation of a turbulent model. We believe that it will be much simpler to integrate SWR techniques into the elsA platform through the use of independent modules. Masking techniques and adaptation to chimera methods are functionalities complementary to the existing ones.

Of course, the use of multi-GPUs can considerably improve the efficiency.

Appendix A

Computation of diffusive Jacobians

The aim of this annexe is to present the main steps in the calculation of the exact expression of the convective and diffusive Jacobians $A, A^v, A^v_{\partial x}, A^v_{\partial y}, B, B^v, B^v_{\partial x}$ and $B^v_{\partial y}$. They are identified in the computation of the the differentials of the viscous fluxes, $F_{vis} + G_{vis}$. Let us recall the expression of the viscous fluxes.

$$F_{vis} + G_{vis} = \begin{pmatrix} 0 \\ -\tau \\ -\tau\vec{u} + q \end{pmatrix},$$

where $\tau = \lambda(\nabla \cdot \vec{u})I + 2\mu D$ is the diffusive tensor. To compute the differentials of the diffusive fluxes we calculate first the differentials of $\nabla \cdot \tau$ et $\nabla \cdot (\tau\vec{u} - q)$. For the simplification of the calculations we introduce another set of notations.

$$\begin{aligned} K(\rho, \rho\vec{u}, \rho E) &= K(\rho, \vec{U}, \tilde{E}) = \nabla \cdot \tau = \nabla \cdot \left[\lambda div(\tfrac{\vec{U}}{\rho})I + \mu\left(\nabla(\tfrac{\vec{U}}{\rho}) + \nabla^T(\tfrac{\vec{U}}{\rho})\right)\right], \\ R(\rho, \rho\vec{u}, \rho E) &= R(\rho, \vec{U}, \tilde{E}) = \nabla \cdot (\tau\vec{u} - q) = \nabla \cdot \left(\tau\tfrac{\vec{U}}{\rho} + \rho\right). \end{aligned}$$

Differential of $\nabla \cdot \tau$ (K)

We proceed to a Taylor development of K in $(\rho + \delta\rho, \vec{U} + \delta\vec{U}, \tilde{E} + \delta\tilde{E})$:

$$\begin{aligned} K(\rho + \delta\rho, \vec{U} + \delta\vec{U}, \tilde{E} + \delta\tilde{E}) = \nabla \cdot \Big[\lambda\nabla \cdot \big(\frac{\vec{U} + \delta\vec{U}}{\rho}(1 - \frac{\delta\rho}{\rho} + O(\delta\rho^2)\big)I + \mu\big(\nabla(\frac{\vec{U} + \delta\vec{U}}{\rho}(1 \\ - \frac{\delta\rho}{\rho} + O(\delta\rho^2)\big) + \nabla^T\big(\frac{\vec{U} + \delta\vec{U}}{\rho}(1 - \frac{\delta\rho}{\rho} + O(\delta\rho^2))\big)\Big] \\ = K(\rho, \vec{U}, \tilde{E}) + \nabla \cdot \big[\lambda div(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})I + \mu\big(\nabla(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2}) \\ + \nabla^T(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})\big)\big] + \nabla \cdot \big[O(\delta\vec{U}\delta\rho) + O(\delta\rho^2)\big]. \end{aligned}$$

The differential of K is the linear part from the Taylor development:

$$\begin{aligned} K'(\rho, \vec{U}, \tilde{E}) \cdot (\delta\rho, \delta\vec{U}, \delta\tilde{E}) = \nabla \cdot \big[\lambda\nabla \cdot (\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})I + \mu\big(\nabla(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2}) \\ + \nabla^T(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})\big)\big] \\ = \nabla \cdot [\tau'(\rho, \vec{U}, \tilde{E}) \cdot (\delta\rho, \delta\vec{U}, \delta\tilde{E})]. \end{aligned} \tag{A.1}$$

Differential of $\nabla \cdot (\tau u - q)$ (R)

Similar to the differential of $\nabla \cdot \tau$ we find the linear part of the development of R in $(\rho + \delta\rho, \vec{U} + \delta\vec{U}, \tilde{E} + \delta\tilde{E})$:

$$\begin{aligned}
R(\rho, \vec{U}, \tilde{E}) &= \nabla \cdot \big[\big(\lambda div(\frac{\vec{U}}{\rho})I + \mu(\nabla\big(\frac{\vec{U}}{\rho}\big) + \nabla^T(\frac{\vec{U}}{\rho}))\big)\frac{\vec{U}}{\rho} + k\nabla(\frac{\tilde{E}}{\rho} - \frac{||\vec{U}||^2}{2\rho^2})\big] \\
R(\rho + \delta\rho, \vec{U} + \delta\vec{U}, \tilde{E} + \delta\tilde{E}) &= R(\rho, \vec{U}, \tilde{E}) + \nabla \cdot \big[(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})\big(\lambda div(\frac{\vec{U}}{\rho})I + \mu(\nabla(\frac{\vec{U}}{\rho}) \\
&\quad + \nabla^T(\frac{\vec{U}}{\rho}))\big)\big] + \nabla \cdot \big\{\frac{\vec{U}}{\rho}\big[\lambda div(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})I + \mu\big(\nabla(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2}) \\
&\quad + \nabla^T(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2})\big)\big] + k\nabla\big(\frac{\delta\tilde{E}}{\rho} - \frac{\tilde{E}}{\rho^2}\delta\rho - \frac{||\delta\vec{U}||^2 + 2\vec{U}\delta\vec{U}}{2\rho^2} \\
&\quad + \frac{||\vec{U}||^2}{\rho^3}\delta\rho\big)\big\} + \nabla \cdot [O(\delta\vec{U}\delta\rho + \delta\rho^2 + \delta\vec{U}^2)].
\end{aligned}$$

We identify the linear part and find the differential of R:

$$\begin{aligned}
R'(\rho, \vec{U}, \tilde{E}) \cdot (\delta\rho, \delta\vec{U}, \delta\tilde{E}) = \nabla \cdot \big[&\tau(\rho, \vec{U}, \tilde{E})(\frac{\delta\vec{U}}{\rho} - \vec{U}\frac{\delta\rho}{\rho^2}) + \tau'(\rho, \vec{U}, \tilde{E}) \cdot (\delta\rho, \delta\vec{U}, \delta\tilde{E})\frac{\vec{U}}{\rho} \\
&+ q'(\rho, \vec{U}, \tilde{E}) \cdot (\delta\rho, \delta\vec{U}, \delta\tilde{E})\big].
\end{aligned} \tag{A.2}$$

Both differentials of R and K require the explicit calculation of τ'.

Calculation of τ'

We express τ in its matrix format:

$$\tau = \frac{1}{\rho^2}\begin{pmatrix}
(-2\lambda)(\rho\frac{\partial}{\partial x}U - U\frac{\partial}{\partial x}\rho) + \lambda(\rho\frac{\partial}{\partial y}V - V\frac{\partial}{\partial y}\rho) & \mu(\rho\frac{\partial}{\partial y}U - U\frac{\partial}{\partial y}\rho + \rho\frac{\partial}{\partial x}V - V\frac{\partial}{\partial x}\rho) \\
\mu(\rho\frac{\partial}{\partial y}U - U\frac{\partial}{\partial y}\rho + \rho\frac{\partial}{\partial x}V - V\frac{\partial}{\partial x}\rho) & \lambda(\rho\frac{\partial}{\partial x}U - U\frac{\partial}{\partial x}\rho) + (-2\lambda)(\rho\frac{\partial}{\partial y}V - V\frac{\partial}{\partial y}\rho)
\end{pmatrix},$$

where U and V are the conservative velocities fields ρu and $V = \rho v$. We compute all coefficients of τ' from its expression given by A.1:

$$\begin{aligned}
\tau'(\rho, \vec{U}, \tilde{E}) \cdot (\delta\rho, \delta\vec{U}, \delta\tilde{E}) = &\begin{pmatrix}
\lambda[\frac{\partial}{\partial x}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) + \frac{\partial}{\partial y}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2})] & 0 \\
0 & \lambda[\frac{\partial}{\partial x}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) + \frac{\partial}{\partial y}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2})]
\end{pmatrix} \\
&+\mu\begin{pmatrix}
\frac{\partial}{\partial x}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) & \frac{\partial}{\partial y}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) \\
\frac{\partial}{\partial x}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2}) & \frac{\partial}{\partial y}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2})
\end{pmatrix} + \mu\begin{pmatrix}
\frac{\partial}{\partial x}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) & \frac{\partial}{\partial x}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2}) \\
\frac{\partial}{\partial y}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) & \frac{\partial}{\partial y}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2})
\end{pmatrix}.
\end{aligned}$$

The first coefficient is τ'_{11},

$$\begin{aligned}
\tau'_{11} &= \lambda[\frac{\partial}{\partial x}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) + \frac{\partial}{\partial y}(\frac{\delta V}{\rho} - V\frac{\delta\rho}{\rho^2})] + 2\mu\frac{\partial}{\partial x}(\frac{\delta U}{\rho} - U\frac{\delta\rho}{\rho^2}) \\
&= \frac{-2\lambda}{\rho^2}[(2\frac{U}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}U)\delta\rho - (\frac{\partial}{\partial x}\rho)\delta U + \rho\frac{\partial}{\partial x}\delta U - U\frac{\partial}{\partial x}\delta\rho] \\
&\quad + \frac{\lambda}{\rho^2}[(2\frac{V}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}V)\delta\rho - (\frac{\partial}{\partial y}\rho)\delta V + \rho\frac{\partial}{\partial y}\delta V - V\frac{\partial}{\partial y}\delta\rho]
\end{aligned}$$

and can be rewritten under the following matrix format:

$$\begin{aligned}\tau'_{11} &= \tfrac{\lambda}{\rho^2}\begin{pmatrix} -4\frac{U}{\rho}\frac{\partial}{\partial x}\rho + 2\frac{\partial}{\partial x}U + 2\frac{V}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}V \\ 2\frac{\partial}{\partial x}\rho \\ -\frac{\partial}{\partial y}\rho \\ 0 \end{pmatrix}\begin{pmatrix}\delta\rho\\ \delta U\\ \delta V\\ \delta\tilde{E}\end{pmatrix} \\ &+\tfrac{\lambda}{\rho^2}\begin{pmatrix}2U\\-2\rho\\0\\0\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial x}\delta\rho\\ \frac{\partial}{\partial x}\delta U\\ \frac{\partial}{\partial x}\delta V\\ \frac{\partial}{\partial x}\delta\tilde{E}\end{pmatrix} + \tfrac{\lambda}{\rho^2}\begin{pmatrix}-V\\0\\\rho\\0\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial y}\delta\rho\\ \frac{\partial}{\partial y}\delta U\\ \frac{\partial}{\partial y}\delta V\\ \frac{\partial}{\partial y}\delta\tilde{E}\end{pmatrix}.\end{aligned}$$

The diffusive tensor is symmetric. We have $\tau'_{12} = \tau'_{21}$ where

$$\begin{aligned}\tau'_{12} = &\frac{\mu}{\rho^2}\big[(2\frac{U}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}U)\delta\rho - (\frac{\partial}{\partial y}\rho)\delta U + \rho\frac{\partial}{\partial y}\delta U - U\frac{\partial}{\partial y}\delta\rho\big] \\ &+ \frac{\mu}{\rho^2}\big[(2\frac{V}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}V)\delta\rho - (\frac{\partial}{\partial x}\rho)\delta V + \rho\frac{\partial}{\partial x}\delta V - V\frac{\partial}{\partial x}\delta\rho\big]\end{aligned}$$

and can be rewritten as:

$$\begin{aligned}\tau'_{12} &= \tfrac{\mu}{\rho^2}\begin{pmatrix} 2\frac{U}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}U + 2\frac{V}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}V \\ -\frac{\partial}{\partial y}\rho \\ -\frac{\partial}{\partial x}\rho \\ 0 \end{pmatrix}\begin{pmatrix}\delta\rho\\ \delta U\\ \delta V\\ \delta\tilde{E}\end{pmatrix} \\ &+\tfrac{\mu}{\rho^2}\begin{pmatrix}-V\\0\\\rho\\0\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial x}\delta\rho\\ \frac{\partial}{\partial x}\delta U\\ \frac{\partial}{\partial x}\delta V\\ \frac{\partial}{\partial x}\delta\tilde{E}\end{pmatrix} + \tfrac{\mu}{\rho^2}\begin{pmatrix}-U\\\rho\\0\\0\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial y}\delta\rho\\ \frac{\partial}{\partial y}\delta U\\ \frac{\partial}{\partial y}\delta V\\ \frac{\partial}{\partial y}\delta\tilde{E}\end{pmatrix}.\end{aligned}$$

The last coefficient is τ'_{22}:

$$\begin{aligned}\tau'_{22} = &\frac{\lambda}{\rho^2}\big[(2\frac{U}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}U)\delta\rho - (\frac{\partial}{\partial x}\rho)\delta U + \rho\frac{\partial}{\partial x}\delta U - U\frac{\partial}{\partial x}\delta\rho\big] \\ &+ \frac{-2\lambda}{\rho^2}\big[(2\frac{V}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}V)\delta\rho - (\frac{\partial}{\partial y}\rho)\delta V + \rho\frac{\partial}{\partial y}\delta V - V\frac{\partial}{\partial y}\delta\rho\big]\end{aligned}$$

$$\begin{aligned}\tau'_{22} &= \tfrac{\lambda}{\rho^2}\begin{pmatrix} 2\frac{U}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}U - 4\frac{V}{\rho}\frac{\partial}{\partial y}\rho + 2\frac{\partial}{\partial y}V \\ -\frac{\partial}{\partial x}\rho \\ 2\frac{\partial}{\partial y}\rho \\ 0 \end{pmatrix}\begin{pmatrix}\delta\rho\\ \delta U\\ \delta V\\ \delta\tilde{E}\end{pmatrix} \\ &+\tfrac{\lambda}{\rho^2}\begin{pmatrix}-U\\\rho\\0\\0\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial x}\delta\rho\\ \frac{\partial}{\partial x}\delta U\\ \frac{\partial}{\partial x}\delta V\\ \frac{\partial}{\partial x}\delta\tilde{E}\end{pmatrix} + \tfrac{\lambda}{\rho^2}\begin{pmatrix}2V\\0\\-2\rho\\0\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial y}\delta\rho\\ \frac{\partial}{\partial y}\delta U\\ \frac{\partial}{\partial y}\delta V\\ \frac{\partial}{\partial y}\delta\tilde{E}\end{pmatrix}.\end{aligned}$$

The calculation of τ' provides us with the three firsts lines of the Jacobian matrix (the first line is actually empty). To facilitate the computation, we change the notations, we build the Jacobian matrix from K' and R' and re-split it in a sum of directional Jacobian matrices (in x,

respectively y direction for a 2D computation):

$$\begin{pmatrix} 0 \\ K'(\rho,\vec{U},\tilde{E})\cdot(\delta\rho,\delta\vec{U},\delta\tilde{E}) \\ R'(\rho,\vec{U},\tilde{E})\cdot(\delta\rho,\delta\vec{U},\delta\tilde{E}) \end{pmatrix} = \big(\frac{\partial}{\partial x}A^v\big)\cdot(\delta\rho,\delta\vec{U},\delta\tilde{E}) + \big(\frac{\partial}{\partial x}A^v_{\partial x}\big).\frac{\partial}{\partial x}(\delta\rho,\delta\vec{U},\delta\tilde{E})$$
$$+ \big(\frac{\partial}{\partial x}A^v_{\partial y}\big).\frac{\partial}{\partial y}(\delta\rho,\delta\vec{U},\delta\tilde{E}) \ + \big(\frac{\partial}{\partial y}B^v\big)\cdot(\delta\rho,\delta\vec{U},\delta\tilde{E})$$
$$+ \big(\frac{\partial}{\partial y}B^v_{\partial x}\big).\frac{\partial}{\partial x}(\delta\rho,\delta\vec{U},\delta\tilde{E}) + \big(\frac{\partial}{\partial y}B^v_{\partial y}\big).\frac{\partial}{\partial y}(\delta\rho,\delta\vec{U},\delta\tilde{E}).$$

We can already give an intermediary expression of each one of them:

$$A^v = \frac{1}{\rho^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ \lambda(-4\frac{U}{\rho}\frac{\partial}{\partial x}\rho + 2\frac{\partial}{\partial x}U + 2\frac{V}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}V) & 2\lambda\frac{\partial}{\partial x}\rho & -\lambda\frac{\partial}{\partial y}\rho & 0 \\ \mu(2\frac{U}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}U + 2\frac{V}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}V) & -\mu\frac{\partial}{\partial y}\rho & -\mu\frac{\partial}{\partial x}\rho & 0 \\ A^v_{41} & A^v_{42} & A^v_{43} & A^v_{44} \end{pmatrix},$$

$$A^v_{\partial x} = \frac{1}{\rho^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 2\lambda U & -2\lambda\rho & 0 & 0 \\ -\mu V & 0 & \mu\rho & 0 \\ A^v_{\partial x41} & A^v_{\partial x42} & A^v_{\partial x43} & A^v_{\partial x44} \end{pmatrix},$$

$$A^v_{\partial y} = \frac{1}{\rho^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -\lambda V & 0 & \lambda\rho & 0 \\ -\mu U & \mu\rho & 0 & 0 \\ A^v_{\partial y41} & A^v_{\partial y42} & A^v_{\partial y43} & A^v_{\partial y44} \end{pmatrix},$$

$$B^v = \frac{1}{\rho^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ \mu(2\frac{U}{\rho}\frac{\partial}{\partial y}\rho - \frac{\partial}{\partial y}U + 2\frac{V}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}V) & -\mu\frac{\partial}{\partial y}\rho & -\mu\frac{\partial}{\partial x}\rho & 0 \\ \lambda(2\frac{U}{\rho}\frac{\partial}{\partial x}\rho - \frac{\partial}{\partial x}U - 4\frac{V}{\rho}\frac{\partial}{\partial y}\rho + 2\frac{\partial}{\partial y}V) & -\lambda\frac{\partial}{\partial x}\rho & 2\lambda\frac{\partial}{\partial y}\rho & 0 \\ B^v_{41} & B^v_{42} & B^v_{43} & B^v_{44} \end{pmatrix},$$

$$B^v_{\partial x} = \frac{1}{\rho^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -\mu V & 0 & \mu\rho & 0 \\ -\lambda U & \lambda\rho & 0 & 0 \\ B^v_{\partial x41} & B^v_{\partial x42} & B^v_{\partial x43} & B^v_{\partial x44} \end{pmatrix},$$

$$B^v_{\partial y} = \frac{1}{\rho^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -\mu U & \mu\rho & 0 & 0 \\ 2\lambda V & 0 & -2\lambda\rho & 0 \\ B^v_{\partial y41} & B^v_{\partial y42} & B^v_{\partial y43} & B^v_{\partial y44} \end{pmatrix}.$$

In order to find the missing coefficients we need to express the differential of q.

$$q(\rho,\vec{U},\tilde{E}) = k\nabla(\frac{\tilde{E}}{\rho} - \frac{||\vec{U}||^2}{2\rho^2}),$$
$$q(\rho+\delta\rho,\vec{U}+\delta\vec{U},\tilde{E}+\delta\tilde{E}) = k\nabla(\frac{\tilde{E}}{\rho} - \frac{||\vec{U}||^2}{2\rho^2}) + k\nabla(\frac{\delta\tilde{E}}{\rho} - \frac{\tilde{E}}{\rho^2}\delta\rho - \frac{\vec{U}.\delta\vec{U}}{\rho^2} + \frac{||\vec{U}||^2}{\rho^3}\delta\rho) + O(\delta\rho^2)$$
$$= q(\rho,\vec{U},\tilde{E}) + q'(\rho,\vec{U},\tilde{E})\cdot(\delta\rho,\delta\vec{U},\delta\tilde{E}) + O(\delta\rho^2).$$

We compute q_1' and q_2' and write their matrix form:

$$
\begin{aligned}
q_1' &= k\frac{\partial}{\partial x}(\frac{\delta\tilde{E}}{\rho} - \tilde{E}\frac{\delta\rho}{\rho^2} - \frac{\vec{U}\delta\vec{U}}{\rho^2} + \frac{||\vec{U}||^2}{\rho^3}\delta\rho) \\
&= \frac{k}{\rho^2}\{[-\frac{\partial}{\partial x}\tilde{E} + \frac{2\tilde{E}}{\rho}(\frac{\partial}{\partial x}\rho) + 2\frac{U}{\rho}\frac{\partial}{\partial x}U + 2\frac{V}{\rho}\frac{\partial}{\partial x}V - \frac{3(U^2+V^2)}{\rho^2}\frac{\partial}{\partial x}\rho]\delta\rho \\
&\quad + (-\frac{\partial}{\partial x}U + \frac{2U}{\rho}\frac{\partial}{\partial x}\rho)\delta U + (-\frac{\partial}{\partial x}V + \frac{2V}{\rho}\frac{\partial}{\partial x}\rho)\delta V + (-\frac{\partial}{\partial x}\rho)\delta\tilde{E} \\
&\quad + (\frac{U^2+V^2}{\rho} - \tilde{E})\frac{\partial}{\partial x}\delta\rho - U\frac{\partial}{\partial x}\delta U - V\frac{\partial}{\partial x}\delta V + \rho\frac{\partial}{\partial x}\delta\tilde{E}\},
\end{aligned}
$$

$$
q_1' = \frac{k}{\rho^2}\begin{pmatrix} -\frac{\partial}{\partial x}\tilde{E} + \frac{2\tilde{E}}{\rho}(\frac{\partial}{\partial x}\rho) + 2\frac{U}{\rho}\frac{\partial}{\partial x}U + 2\frac{V}{\rho}\frac{\partial}{\partial x}V - \frac{3(U^2+V^2)}{\rho^2}\frac{\partial}{\partial x}\rho \\ -\frac{\partial}{\partial x}U + \frac{2U}{\rho}\frac{\partial}{\partial x}\rho \\ -\frac{\partial}{\partial x}V + \frac{2V}{\rho}\frac{\partial}{\partial x}\rho \\ -\frac{\partial}{\partial x}\rho \end{pmatrix}\begin{pmatrix}\delta\rho\\ \delta U\\ \delta V\\ \delta\tilde{E}\end{pmatrix}
+\frac{k}{\rho^2}\begin{pmatrix}\frac{U^2+V^2}{\rho} - \tilde{E}\\ -U\\ -V\\ \rho\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial x}\delta\rho\\ \frac{\partial}{\partial x}\delta U\\ \frac{\partial}{\partial x}\delta V\\ \frac{\partial}{\partial x}\delta\tilde{E}\end{pmatrix},
$$

we get a similar result in the y direction $q_2' = k\frac{\partial}{\partial y}(\frac{\delta\tilde{E}}{\rho} - \tilde{E}\frac{\delta\rho}{\rho^2} - \frac{\vec{U}\delta\vec{U}}{\rho^2} + \frac{||\vec{U}||^2}{\rho^3}\delta\rho)$,

$$
q_2' = \frac{k}{\rho^2}\begin{pmatrix} -(\frac{\partial}{\partial y}\tilde{E} - \frac{2\tilde{E}}{\rho}\frac{\partial}{\partial y}\rho) + [2\frac{U}{\rho}\frac{\partial}{\partial y}U + 2\frac{V}{\rho}\frac{\partial}{\partial y}V - \frac{3(U^2+V^2)}{\rho}\frac{\partial}{\partial y}\rho] \\ -\frac{\partial}{\partial y}U + \frac{2U}{\rho}\frac{\partial}{\partial y}\rho \\ -\frac{\partial}{\partial y}V + \frac{2V}{\rho}\frac{\partial}{\partial y}\rho \\ -\frac{\partial}{\partial y}\rho \end{pmatrix}\begin{pmatrix}\delta\rho\\ \delta U\\ \delta V\\ \delta\tilde{E}\end{pmatrix}
+\frac{k}{\rho^2}\begin{pmatrix}\frac{U^2+V^2}{\rho} - \tilde{E}\\ -U\\ -V\\ \rho\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial y}\delta\rho\\ \frac{\partial}{\partial y}\delta U\\ \frac{\partial}{\partial y}\delta V\\ \frac{\partial}{\partial y}\delta\tilde{E}\end{pmatrix}.
$$

Computation of the last line of the Jacobian in $x-$ direction:

$$
\begin{aligned}
A_{41}^v &= -\frac{U}{\rho^2}\tau_{11} - \frac{V}{\rho^2}\tau_{12} + \frac{U}{\rho}\tau_{11}'[0] + \frac{V}{\rho}\tau' 12[0] + q_1'[0] \\
&= \frac{1}{\rho^2}\big[(3(-2\lambda-k)\frac{U^2}{\rho^2} + 3(\mu-k)\frac{V^2}{\rho^2} + 2k\frac{\tilde{E}}{\rho})\frac{\partial}{\partial x}\rho + \mu\frac{UV}{\rho^2}\frac{\partial}{\partial y}\rho \\
&\quad + 2(k+2\lambda)\frac{U}{\rho}\frac{\partial}{\partial x}U - 2\mu\frac{V}{\rho}\frac{\partial}{\partial y}U + 2(k-\mu)\frac{V}{\rho}\frac{\partial}{\partial x}V - 2\lambda\frac{U}{\rho}\frac{\partial}{\partial y}V - k\frac{\partial}{\partial x}\tilde{E}\big], \\
A_{\partial x41}^v &= \frac{U}{\rho}\tau_{11}'[0] + \frac{V}{\rho}\tau' 12[0] + q_1'[0] = \frac{1}{\rho^2}(\frac{2\lambda+k}{\rho}U^2 + \frac{-\mu+k}{\rho}V^2 - k\tilde{E}), \\
A_{\partial y41}^v &= \frac{U}{\rho}\tau_{11}'[0] + \frac{V}{\rho}\tau' 12[0] + q_1'[0] = \frac{\lambda}{2\rho^3}UV, \\
A_{42}^v &= \frac{1}{\rho}\tau_{11} + \frac{1}{\rho}\tau_{12} + \frac{U}{\rho}\tau_{11}'[1] + \frac{V}{\rho}\tau' 12[1] + q_1'[1] \\
&= \frac{1}{\rho^2}\big[((2k - \frac{8}{3}\mu)\frac{U}{\rho} - \mu\frac{V}{\rho})\frac{\partial}{\partial x}\rho + (-\mu\frac{U}{\rho} - \frac{1}{3}\mu\frac{V}{\rho})\frac{\partial}{\partial y}\rho \\
&\quad + (-k + \frac{4}{3}\mu)\frac{\partial}{\partial x}U + \mu\frac{\partial}{\partial y}U + \mu\frac{\partial}{\partial x}V - \frac{2}{3}\mu\frac{\partial}{\partial y}V\big],
\end{aligned}
$$

$$
\begin{aligned}
A^v_{\partial x42} &= \frac{U}{\rho}\tau'_{11}[1] + \frac{V}{\rho}\tau' 12[1] + q'_1[1] = \frac{1}{\rho^2}(-2\lambda - k)U,\\
A^v_{\partial y42} &= \frac{U}{\rho}\tau'_{11}[1] + \frac{V}{\rho}\tau' 12[1] + q'_1[1] = \frac{\mu}{\rho^2}V,\\
A^v_{43} &= \frac{1}{\rho}\tau_{11} + \frac{1}{\rho}\tau_{12} + \frac{U}{\rho}\tau'_{11}[2] + \frac{V}{\rho}\tau' 12[2] + q'_1[2]\\
&= \frac{1}{\rho^2}\big[\big(2(k-\mu)\frac{V}{\rho} - \frac{4}{3}\mu\frac{U}{\rho}\big)\frac{\partial}{\partial x}\rho + (-\frac{1}{3}\mu\frac{U}{\rho} + \frac{2}{3}\mu\frac{V}{\rho})\frac{\partial}{\partial y}\rho\\
&\quad + \frac{4}{3}\mu\frac{\partial}{\partial x}U + \mu\frac{\partial}{\partial y}U + (\mu - k)\frac{\partial}{\partial x}V - \frac{2}{3}\mu\frac{\partial}{\partial y}V\big],\\
A^v_{\partial x43} &= \frac{U}{\rho}\tau'_{11}[2] + \frac{V}{\rho}\tau' 12[2] + q'_1[2] = \frac{\mu - k}{\rho^2}V,\\
A^v_{\partial y43} &= \frac{U}{\rho}\tau'_{11}[2] + \frac{V}{\rho}\tau' 12[2] + q'_1[2] = \frac{\lambda}{\rho^2}U,\\
A^v_{44} &= \frac{U}{\rho}\tau'_{11}[3] + \frac{V}{\rho}\tau' 12[3] + q'_1[3] = -\frac{k}{\rho^2}\frac{\partial}{\partial x}\rho,
\end{aligned}
$$

$$
\begin{aligned}
A^v_{\partial x44} &= \frac{U}{\rho}\tau'_{11}[3] + \frac{V}{\rho}\tau' 12[3] + q'_1[3] = \frac{k}{\rho^2}\rho,\\
A^v_{\partial y44} &= \frac{U}{\rho}\tau'_{11}[3] + \frac{V}{\rho}\tau' 12[3] + q'_1[3] = 0.
\end{aligned}
$$

Computation of the last line of the Jacobian matrix in $y-$direction:

$$
\begin{aligned}
B^v_{41} &= -\frac{U}{\rho^2}\tau_{21} - \frac{V}{\rho^2}\tau_{22} + \frac{U}{\rho}\tau'_{21}[0] + \frac{V}{\rho}\tau'_{22}[0] + q'_2[0]\\
&= \frac{1}{\rho^2}\big[\mu\frac{UV}{\rho^2}\frac{\partial}{\partial x}\rho + \big(3(\mu - k)\frac{U^2}{\rho^2} + 3(-2\lambda - k)\frac{V^2}{\rho^2} + 2k\frac{\tilde{E}}{\rho}\big)\frac{\partial}{\partial y}\rho\\
&\quad - 2\lambda\frac{V}{\rho}\frac{\partial}{\partial x}U + 2(k - \mu)\frac{U}{\rho}\frac{\partial}{\partial y}U - 2\mu\frac{U}{\rho}\frac{\partial}{\partial x}V + 2(k + 2\lambda)\frac{V}{\rho}\frac{\partial}{\partial y}V - k\frac{\partial}{\partial y}\tilde{E}\big],\\
B^v_{\partial x41} &= \frac{U}{\rho}\tau'_{21}[0] + \frac{V}{\rho}\tau' 22[0] + q'_2[0] = -\frac{1}{3}\frac{\mu}{\rho^2}UV,\\
B^v_{\partial y41} &= \frac{U}{\rho}\tau'_{21}[0] + \frac{V}{\rho}\tau' 22[0] + q'_2[0] = \frac{1}{\rho^2}\big(\frac{k - \mu}{\rho}U^2 + \frac{2\lambda + k}{\rho}V^2 - k\tilde{E}\big),
\end{aligned}
$$

$$
\begin{aligned}
B^v_{42} &= \frac{1}{\rho}\tau_{21} + \frac{1}{\rho}\tau_{22} + \frac{U}{\rho}\tau'_{21}[1] + \frac{V}{\rho}\tau' 22[1] + q'_2[1]\\
&= \frac{1}{\rho^2}\big[-\frac{\mu}{3}(\frac{U}{\rho} + \frac{V}{\rho})\frac{\partial}{\partial x}\rho + \big(2(k - \mu)\frac{U}{\rho} - \frac{4}{3}\mu\frac{V}{\rho}\big)\frac{\partial}{\partial y}\rho\\
&\quad - \frac{2}{3}\mu\frac{\partial}{\partial x}U + (\mu - k)\frac{\partial}{\partial y}U + \mu\frac{\partial}{\partial x}V + \frac{4}{3}\mu\frac{\partial}{\partial y}V\big],\\
B^v_{\partial x42} &= \frac{U}{\rho}\tau'_{21}[1] + \frac{V}{\rho}\tau' 22[1] + q'_2[1] = \frac{\lambda}{\rho^2}V,\\
B^v_{\partial y42} &= \frac{U}{\rho}\tau'_{21}[1] + \frac{V}{\rho}\tau' 22[1] + q'_2[1] = \frac{\mu - k}{\rho^2}U,\\
B^v_{43} &= \frac{1}{\rho}\tau_{21} + \frac{1}{\rho}\tau_{22} + \frac{U}{\rho}\tau'_{21}[2] + \frac{V}{\rho}\tau'_{22}[2] + q'_2[2]\\
&= \frac{1}{\rho^2}\big[(-\frac{1}{3}\mu\frac{U}{\rho} - \mu\frac{V}{\rho})\frac{\partial}{\partial x}\rho + \big(-\mu\frac{U}{\rho} + 2(k - \frac{4}{3}\mu)\frac{V}{\rho}\big)\frac{\partial}{\partial y}\rho\\
&\quad - \frac{4}{3}\mu\frac{\partial}{\partial x}U + \mu\frac{\partial}{\partial y}U + \mu\frac{\partial}{\partial x}V + (\frac{4}{3}\mu - k)\frac{\partial}{\partial y}V\big],
\end{aligned}
$$

$$
\begin{aligned}
B^v_{\partial x43} &= \frac{U}{\rho}\tau'_{21}[2] + \frac{V}{\rho}\tau' 22[2] + q'_2[2] = \frac{\mu}{\rho^2}U,\\
B^v_{\partial y43} &= \frac{U}{\rho}\tau'_{21}[2] + \frac{V}{\rho}\tau' 22[2] + q'_2[2] = \frac{-2\lambda - k}{\rho^2}V,\\
B^v_{44} &= \frac{U}{\rho}\tau'_{21}[3] + \frac{V}{\rho}\tau' 22[3] + q'_2[3] = -\frac{k}{\rho^2}\frac{\partial}{\partial y}\rho,\\
B^v_{\partial x44} &= \frac{U}{\rho}\tau'_{21}[3] + \frac{V}{\rho}\tau' 22[3] + q'_2[3] = 0,\\
B^v_{\partial y44} &= \frac{U}{\rho}\tau'_{21}[3] + \frac{V}{\rho}\tau' 22[3] + q'_2[3] = \frac{k}{\rho^2}\rho.
\end{aligned}
$$

Calculation of the eigenvalues

Let J be a matrix of common form of $A^v, A^v_{\partial x}, A^v_{\partial y}, B^v, B^v_{\partial x}$ and $B^v_{\partial y}$:

$$
J = \begin{pmatrix} 0 & 0 & 0 & 0 \\ J_{21} & J_{22} & J_{23} & 0 \\ J_{31} & J_{32} & J_{33} & 0 \\ J_{41} & J_{42} & J_{43} & J_{44} \end{pmatrix}
$$

We search λ such us $|J - \lambda I| = 0$. We find four different relations:

$$
\begin{aligned}
&\lambda_0 = 0,\\
&\lambda_1 = J_{44},\\
&\lambda_2 = \tfrac{J_{22}+J_{33}-\sqrt{\Delta}}{2},\\
&\lambda_3 = \tfrac{J_{22}+J_{33}+\sqrt{\Delta}}{2}
\end{aligned}
$$

where $\Delta = J^2_{22} + J^2_{33} - 2J_{22}J_{33} + 4J_{32}J_{23}$. The numerical computation also requires the values of ρ_J, the spectral of the Jacobian matrix, $\rho_J = \max(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$.

For A^v we have:

$$
J_{22} = -\tfrac{\mu}{\rho^2}\tfrac{4}{3}\tfrac{\partial}{\partial x}\rho, \quad J_{23} = \tfrac{\mu}{\rho^2}\tfrac{2}{3}\tfrac{\partial}{\partial y}\rho, \quad J_{32} = -\tfrac{\mu}{\rho^2}\tfrac{\partial}{\partial y}\rho, \quad J_{33} = -\tfrac{\mu}{\rho^2}\tfrac{\partial}{\partial x}\rho, \quad J_{44} = -\tfrac{k}{\rho^2}\tfrac{\partial}{\partial x}\rho
$$

$$
\text{and } \Delta = J^2_{22} + J^2_{33} - 2J_{22}J_{33} + 4J_{32}J_{23} = \frac{\mu^2}{\rho^4}\big[\frac{1}{9}(\frac{\partial}{\partial x}\rho)^2 - \frac{8}{3}(\frac{\partial}{\partial y}\rho)^2\big].
$$

We note $\Delta = \frac{\mu^2}{\rho^4}\Delta'$ and find the following eigenvalues:

$$
\begin{cases}
\lambda_0 = 0,\\
\lambda_1 = -\frac{k}{\rho^2}\frac{\partial}{\partial x}\rho,\\
\lambda_2 = \frac{\mu}{2\rho^2}(-\frac{7}{3}\frac{\partial}{\partial x}\rho - \sqrt{\Delta'}),\\
\lambda_3 = \frac{\mu}{2\rho^2}(-\frac{7}{3}\frac{\partial}{\partial x}\rho + \sqrt{\Delta'}).
\end{cases}
$$

For $A^v_{\partial x}$ we

find the following eigenvalues: $\lambda_0 = 0,\ \lambda_1 = \frac{k}{\rho},\ \lambda_2 = \frac{7\mu}{3\rho},\ \lambda_3 = 0.$

For $A^v_{\partial y}$ all eigenvalues equal zero: $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = 0$.

B^v has the same form as J with:

$$
J_{22} = -\tfrac{\mu}{\rho^2}\tfrac{\partial}{\partial y}\rho, \quad J_{23} = -\tfrac{\mu}{\rho^2}\tfrac{\partial}{\partial x}\rho, \quad J_{32} = -\tfrac{\lambda}{\rho^2}\tfrac{\partial}{\partial x}\rho, \quad J_{33} = \tfrac{2\lambda}{\rho^2}\tfrac{\partial}{\partial y}\rho, \quad J_{44} = -\tfrac{k}{\rho^2}\tfrac{\partial}{\partial y}\rho
$$

and $\Delta = \frac{\mu^2}{\rho^4}\big[\frac{1}{9}(\frac{\partial}{\partial y}\rho)^2 - \frac{8}{3}(\frac{\partial}{\partial x}\rho)^2\big]$. We note $\Delta = \frac{\mu^2}{\rho^4}\Delta'$ and find the following eigenvalues:

$$
\begin{cases}
\lambda_0 = 0,\\
\lambda_1 = -\frac{k}{\rho^2}\frac{\partial}{\partial y}\rho,\\
\lambda_2 = \frac{\mu}{2\rho^2}(-\frac{7}{3}\frac{\partial}{\partial y}\rho - \sqrt{\Delta'}),\\
\lambda_3 = \frac{\mu}{2\rho^2}(-\frac{7}{3}\frac{\partial}{\partial y}\rho + \sqrt{\Delta'}).
\end{cases}
$$

For $B^v_{\partial x}$ we find eigenvalues equal to zero: $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = 0$.
For $B^v_{\partial y}$ we find the following eigenvalues: $\lambda_0 = 0,\ \lambda_1 = \frac{k}{\rho},\ \lambda_2 = \frac{4\mu}{3\rho},\ \lambda_3 = \frac{\mu}{\rho}$.

Appendix B

List of GPU libraries

LibGeoDecomp

- a library for geometric decomposition codes based on C++ templates,
- automatically parallelises regular 2D/3D grids,
- home page: *http://www.libgeodecomp.org/*.

MAGMA

- stands for Matrix Algebra on GPU and Multicore Architectures,
- it is based on C++ STL extensions,
- it is a hybrid library between LAPACK/ScaLAPACK and Tile Algorithms for sparse linear algebra,
- works on CUDA, Intel Xeon Phi and OpenCL and has a multiGPU support,
- home page: *http://icl.cs.utk.edu/magma*.

Thrust

- stands for Standard Template Library for GPUs,
- contains functions to easy manipulate vectors on GPUs (reductions, sorting and other operators),
- written in C++ it is based on CUDA, thus working only on NVIDIA GPUs.
- home page: *http://thrust.github.io/*.

CuSP and CUBLAS

- Cuda SPARSE was developed by NVIDIA on top of Thrust and works only for NVIDIA GPUs,
- is able to solve direct methods for linear solver (QR factorisation, GMRES solver), preconditioned Iterative methods, incomplete-LU factorisations, sparse matrix-vector multiplications,
- home page: *https://github.com/cusplibrary*.

ViennaCL

- free opensource sparse linear algebra library, it is an extension of Boost uBLAS library,
- it is written in C++ and works on multiple platforms: OpenMP, OpenCL, CUDA,
- has BLAS 1-3 support,
- home page: *http://viennacl.sourceforge.net/*.

Paralution

- open-source library for sparse linear algebra on multiple platforms (OpenMP, OpenCL, CUDA),
- contains sparse iterative solvers (CR, CG, BICGStab, GMRES, IDR), multigrid (CMG, AMG), Deflated PCG, Fixed-point iteration schemes, Mixed-precision schemes and fine-grained parallel preconditioners based on splitting, ILU factorisation with levels, additive Schwarz and approximate inverse.
- item home page: *http://www.paralution.com/*.

PETSc

- supports linear and non-linear PDE problems on structured and unstructured meshes, Krylov methods: Conjugate Gradien, GMRES, CG-Squared, Bi-CG-stab, Transpose-free QMR, etc., preconditioners: Block Jacobi, overlapping additive Schwarz ICC, ILU, LU, etc.,
- supports MPI, GPU, can invoke CuSP and ViennaCL if needed,
- home page: *http://www.mcs.anl.gov/petsc* .

Trilinos

- it is a collection of packages for solving large-scale, complex multi-physics problems,
- supports basic linear algebra, preconditioners, iterative linear-solvers, direct linear solvers (SuperLU, ScaLAPACK, MUMPS,...), non-linear solvers (NOX,...), eigensolvers, mesh generation and adaptivity, partitioning, etc.,
- home page: *http://trilinos.org/*.

Appendix C

Example of CUDA programming. Minimum reduction

We present here the entire code to compute a global minimum reduction inside a device. The aim is to highlight the difficulties of programming on the GPU device, the need for a manual treatment of each level of hierarchy. On GPU (CUDA language), the algorithm to compute the time step inside one cell is equivalent to the one on CPU (Fortran/C/C++).

```
// ------------------------------------------------------
template<typename K> __device__ K
gpu_compute_dt(int dir,K gamma,K mu, K Pr, K cfl, K dx,  K rho, K rhoU, K
    rhoV, K rhoW, K rhoE)
{
  K rho_i, u[3], p, a, dtC,dtD;
  rho_i = 1./rho;
  u[0]  = rho_i*rhoU;
  u[1]  = rho_i*rhoV;
  u[2]  = rho_i*rhoW;
  p     = (gamma-1.)*(rhoE-0.5*rho*(u[0]*u[0]+u[1]*u[1]+u[2]*u[2]));
  a     = sqrt(gamma*p*rho_i);

  dtC   = cfl*dx/(fabs(u[dir])+a) ;
  dtD   = 0.5*cfl*dx*dx*rho*Pr/(mu*gamma);

  return gpu_min(dtC,dtD);
}
```

The minimum reduction algorithm is necessary on the device (only optional on the host). The reduction must be done inside each level of device hierarchy. We present the framework set done by the host, then the kernel and local to device functions used to compute a minimum over a 2D computational domain.

```
/* ------------------------------------------------------
 CPU program
 compute the local time step for a  2D domain using a directional
    treatment: */
template<> double
Zone<double>::compute_reg_LocalTimeStep
               (int nci,int ncj,int nck,
```

```
                int order, double dx, double dy,
                Array<double>& rho,
                Array<double>& rhoU, Array<double>& rhoV,
                Array<double>& rhoW, Array<double>& rhoE )
{

  int nb_blockx = (nci+31)/32, nb_blocky = (ncj-2*order+15)/16;
  int i,n=1,nb_block = nb_blockx*nb_blocky;
  double* dt0 = new double[2*nb_block];
  for (i=0;i<2*nb_block;i++)
    dt0[i] = 100.;
  int dimDt = 2*nb_block;
  Array<double> min_dt(dimDt);
  min_dt.setDataFromHost(dt0);

  /* compute minimum time step
     x direction
     2D framwork set: set the size of grids and blocks inside the device
         */
  dim3 threads(16,16,1);  // 2D block
  dim3 grid(nb_blockx,nb_blocky, 1); // 2D grid
  /* call of the kernel */
  gpu_compute_reg_BlockTimeStep<double,16>
              <<<grid,threads>>>(min_dt.getGpuData(),0,
              m_pt_eos->getGamma(),
              m_pt_eos->getMu(), m_pt_eos->getPr(),
              m_cfl, order, nci, ncj, rho.stride()/sizeof(double),
              dx,
              rho.getGpuData(), rhoU.getGpuData(),
              rhoV.getGpuData(),rhoW.getGpuData(),
              rhoE.getGpuData());
  /* y direction
     call of the kernel */
   gpu_compute_reg_BlockTimeStep<double,16>
              <<<grid,threads>>>((min_dt.getGpuData()+nb_block),
              1, m_pt_eos->getGamma(),
              m_pt_eos->getMu(), m_pt_eos->getPr(),
              m_cfl,
              order, nci, ncj, rho.stride()/sizeof(double), dy,
              rho.getGpuData(), rhoU.getGpuData(),
              rhoV.getGpuData(),rhoW.getGpuData(),
              rhoE.getGpuData());

  /* compute a 1D minimum reduction for an array containing the resulting
       minimum array in the x direction concatenated
     with the resulting minimum array in the y direction */
  nb_block = 2*nb_block;
  while (nb_block>512){
    /* framwork set: set the size of grids and blocks inside the device
        */
    nb_block = (nb_block+255)/256;
    dim3 threads2(256,1,1);     // 1D block
    dim3 grid2(nb_block,1, 1); // 1D grid
    /* call of the kernel */
    gpu_reduce_min<double,256><<<grid2,threads2>>>(n,dimDt,min_dt.
        getGpuData(),min_dt.getGpuData());

  }
  double* dtMins = new double[nb_block];
  /* copy  data from the device */
  min_dt.getDataFromDevice(dtMins,nb_block);
```

```
  double minDt = dtMins[0];
  for (unsigned i = 1; i < nb_block; ++i)
    minDt = std::min(minDt, dtMins[i]);
  delete [] dtMins;
  return minDt;
}
}

/* ----------------------------------------------------------
   Computation of the global time step using a 2D reduction method to
       find the global minimum value
   Generalisation for different block size */
template<typename K, int blockSize> __global__ void
gpu_compute_reg_BlockTimeStep( K* min_odt, int dir,
                               K gamma, K mu, K Pr,K cfl,int order,
                               int nx, int ny, int stride, K dx,
                               K* rho, K* rhoU, K* rhoV, K* rhoW, K* rhoE
                                    )
{
  __shared__ K min_dt[blockSize*blockSize];

  //each thread loads one element from global to shared memory
  unsigned int tid = threadIdx.x + threadIdx.y*blockDim.x;
  unsigned int i   = threadIdx.x + blockSize*(blockIdx.x*2);
  unsigned int j   = (threadIdx.y+order) + blockDim.y*blockIdx.y;
  unsigned int ind = i  + j*stride;
  min_dt[tid]      = 1E6;
  if (j < ny-order) {
    K dt1,dt2;
    // compute min dt in each of two  directions
    dt1 = (i > order-1 && i < nx-order) ?
          gpu_compute_dt (dir,gamma,mu,Pr,cfl,dx,
                          rho[ind],rhoU[ind],rhoV[ind],
                          rhoW[ind],rhoE[ind]) : min_dt[tid];
    dt2 = (i+blockSize < nx-order) ?
          gpu_compute_dt(dir,gamma,mu,Pr,cfl,dx,
                         rho[ind+blockSize] ,rhoU[ind+blockSize],
                         rhoV[ind+blockSize],rhoW[ind+blockSize],
                         rhoE[ind+blockSize]) : min_dt[tid];
    min_dt[tid] = gpu_min(min_dt[tid],dt1);
    min_dt[tid] = gpu_min(min_dt[tid],dt2);
    }
  /* synchronisation inside a block  */
  __syncthreads();

  if (blockSize*blockSize>= 1024){
    if (tid < 512)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+512]);
    __syncthreads();
  }
  if (blockSize*blockSize>= 512){
    if (tid < 256)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+256]);
    __syncthreads();
    }
  if (blockSize*blockSize>= 256){
    if (tid < 128)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+128]);
    __syncthreads();
  }
  if (blockSize*blockSize>= 128){
```

```
    if (tid < 64)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+ 64]);
    __syncthreads();
  }

  if (tid<32)
    gpu_warpReduce<K,blockSize*blockSize>(min_dt, tid);
  __syncthreads();
  if (tid==0)
    min_odt[blockIdx.x+blockIdx.y*gridDim.x] = min_dt[0];
}


/* ---------------------------------------------------------
   1D reduction
   reduces the number of blocks to its half.
   for each two blocks  we reduce the minimum of the vector min_idt (
       global memory) and save the result in min_odt (global memory)
   the function take into account different block size depending on the
       GPU generation
   kernel function: accessible by the CPU */
template<typename K, int blockSize> __global__ void
gpu_reduce_min(unsigned int n, int dimDt,  K* min_idt, K* min_odt)
{
  __shared__ K min_dt[blockSize];
  /* each thread loads one element from global to shared memory */
  unsigned int tid = threadIdx.x;
  unsigned int i   = threadIdx.x + blockSize*(blockIdx.x*2);

  min_dt[tid]      = 1E6;

  if (i<dimDt)
    min_dt[tid] = (i+blockSize < dimDt) ? gpu_min( gpu_min(min_dt[tid],
        min_idt[i]), min_idt[i+blockSize]):
      gpu_min(min_dt[tid],min_idt[i]);
  __syncthreads();

  if (blockSize>= 1024){
    if (tid < 512)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+512]);
    __syncthreads();
  }
  if (blockSize>= 512){
    if (tid < 256)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+256]);
    __syncthreads();
  }
  if (blockSize>= 256){
    if (tid < 128)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+128]);
    __syncthreads();
  }
  if (blockSize>= 128){
    if (tid < 64)
      min_dt[tid] =  gpu_min(min_dt[tid],min_dt[tid+ 64]);
    __syncthreads();
  }
  if (tid<32) gpu_warpReduce<K,blockSize>(min_dt, tid);
  if (tid==0) min_odt[blockIdx.x] = min_dt[0];
}
```

```
/* ------------------------------------------------------------
   reduction of the last block -> reduction by warps -> unroll of the
       last warp, a warp contains 32 threads
   at the end of this reduction one thread will contain the minimum over
       the entire block */
template<typename K, int blockSize> __device__ void
gpu_warpReduce(volatile K* min_dt, unsigned int tid)
{
  if (blockSize >= 64)
    min_dt[tid] = gpu_min(min_dt[tid], min_dt[tid+32]);
  if (blockSize >= 32)
    min_dt[tid] = gpu_min(min_dt[tid], min_dt[tid+16]);
  if (blockSize >= 16)
    min_dt[tid] = gpu_min(min_dt[tid], min_dt[tid+ 8]);
  if (blockSize >=  8)
    min_dt[tid] = gpu_min(min_dt[tid], min_dt[tid+ 4]);
  if (blockSize >=  4)
    min_dt[tid] = gpu_min(min_dt[tid], min_dt[tid+ 2]);
  if (blockSize >=  2)
    min_dt[tid] = gpu_min(min_dt[tid], min_dt[tid+ 1]);
}


// classical minimum computation, necessary to compute minimum between
    two values stored inside two different threads
// visible only by the device, can be done between float or double values
     (K in {float, double})
template<typename K> __device__ K
gpu_min(K a, K b)
{
  if (a>0 and b>0)
    return (a<b ? a : b);
  else
    return 100;
}
```

Nomenclature

General Notations

Φ	finite volume test function
Ψ	total flux (convective and diffusive)
Ψ_{conv}	total convective flux
Ψ_{vis}	total viscous flux
ρ	the mass density
τ	the viscous tensor of constraints
$\top$	the absolute temperature
$\vec{u} = (u, v, w)^t$	the velocity vector
D	the deformation tensor
E	the total energy per unit
e	the internal energy per unit mass
F_{Euler}	convective or Euler flux in x direction
F_{vis}	viscous or diffusive flux in x direction
G_{Euler}	convective or Euler flux in y direction
G_{vis}	viscous or diffusive flux in y direction
H_{Euler}	convective or Euler flux in z direction
H_{vis}	viscous or diffusive flux in z direction
p	the fluid pressure
q	the heat flux
u	the x-component of velocity
$U = (\rho, \rho\vec{u}, \rho E)$	the vector of conservative values
v	the y-component of velocity
w	the z-component of velocity

Constants Symbols

γ	the ratio of specific heats
λ	the viscosity coefficient of the fluid
μ	the viscosity coefficient of the fluid

c_p constant pressure specific heat coefficient

c_v constant volume specific heat coefficient

$K_\top$ the conductivity coefficient

M_a Mach number

Pr the Prandtl number (assumed to be constant)

r_{gas} the ratio of the universal constant of perfect gases to the molar mass of the considered gas

Re Reynolds number

Mesh related Symbols

Γ interface of one cell

ν_{Ω_i} volume of the cell Ω_i

Ω arbitrary domain of definition

$\partial\Omega$ boundary of Ω

i, j, l, k indexes

N number of non-overlapping cells

n_{Ω_i} unit outer normal to the cell Ω_i

T the time interval limit

t time instant

Bibliography

[1] Ascher, U.M., Petzold, L.R.: Computer Mthods for Ordinary Differential Equations ans Differential-Algebraic Equations. Philadelphia, PA: SIAM Press (1998)

[2] Audusse, E., Dreyfuss, P., Merlet, B.: Optimized schwarz Waveform Relaxation for the Primitive equations of the ocean. SIAM Journal on Scientific Computing pp. 2908–2936 (2010)

[3] Bal, G., Maday, Y.: A Parareal Time Discretization for Non-Linear PDE's with Application to the Pricing of an American Put. Lectures Notes in Computational Science and Engineering **23**, 189–202 (2002)

[4] Beam, R.M., Warming, R.: An Implicit Factored Scheme for the Compressible Navier–Stokes Equations. AIAA Journal **16**(4), 393–402 (1978)

[5] Bjorhus, M.: On dynamic iteration for delay differential equations. BIT Journal, Biosystems and Information Technology **34**(3), 325–336 (1994)

[6] Bjorhus, M.: Semi-discrete subdomain iteration for hyperbolic systems. The Norvegian Institut of Technology (1995)

[7] Blayo, E., Halpern, L., Japhet, C.: Optimized Schwarz waveform relaxation algorithms with nonconforming time discretization for coupling convection-diffusion problems with discontinuous coefficients. Domain Decomposition Methods in Science ans Engineering **16**, 267–274 (2007)

[8] Borrel, M., Halpern, L., Ryan, J.: Euler/Navier-Stokes couplings for multiscale aeroacoustic problems. AIAA Computational Fluid Dynamics pp. 427–433 (2011)

[9] Bouris, D., Bergeles, G.: 2D LES of vortex shedding from a square cylinder. Journal of Wind Engineering and Industrial Aerodynamics **80**, 31–46 (1999)

[10] Brezzi, F., Lions, J.L., Pironneau, O.: Analysis of a Chimera method. Comptes rendus de l'Académie des sciences. Série 1, Mathématique **332**(7), 655–660 (2001)

[11] Burrage, K.: Parallel methods for systems of ordinary differential equations. SIAM News **26**(5) (1995)

[12] Burrage, K., Dyke, C.: On the performance of parallel waveform relaxations for differential systems. Lexington International Conference on the Method of Lines (1995)

[13] Cai, X.: Overlapping domain decomposition methods. Advanced Topics in Computational Partial Differential Equations pp. 57–95 (2003)

[14] Cai, X.C., Dryja, M.: Domain Decomposition Methods for Monotone Nonlinear Elliptic Problems. Domain Decomposition Methods in Scientific and Engineering Computing **180**, 21–27 (1994)

[15] Cai, X.C., Gropp, W., Keyes, D.E., Tidriri, M.: Newton-Krylov-Schwarz Methods in cfd. Notes on Numerical Fluid Mechanics **47**, 17–30 (1999)

[16] Cai, X.C., Keyes, D.E.: Nonlinearly Preconditioned Inexact Newton Algorithms. SIAM Journal on Scientific Computing **24**(1), 183–200 (2002)

[17] Cai, X.C., Keyes, D.E., Marcinkowski, L.: Non-linear additive Schwarz precondtioners and application in computational fluid dynamics. International Journal for numerical Methods in Fluids **40**(12), 1463–1470 (2002)

[18] Cai, X.C., Keyes, D.E., Venkatakrishnan, V.: Newton-Krylov-Schwarz: An Implicit Solver for cfd. Technical Report: Institute for Computer Applications in Science and Engineering (1995)

[19] Chen, X., Phoon, K.: Some numerical experiences on convergence criteria for iterative finite element solvers. Computers and Geotechnics **36**, 1272–1284 (2009)

[20] Choquet, R.: Etude de la méthode de Newton -GMRES. Phd thesis, University de Rennes I (1995)

[21] Coakley, T.: Numerical Method for gaz dynamics combining characteristic and conservation concepts. AIAA 14th Fluid and Plasma Dynamics Conference (1981)

[22] Coakley, T.: Implicit Upwind Methods for the Compressible Navier–Stokes Equations. AIAA Journal **23**(3), 374–380 (1985)

[23] Colonius, T.: Modeling Artificial Boundary Conditiond for Compressible Flow. Annu. Rev. Fluid Mech. **36** (2004)

[24] Colonius, T., Lele, S.K., Moin, P.: Sound generation in a mixing layer. Journal of Fluid Mechanics **330**, 375–409 (1997)

[25] CUDA: Home page. http://www.nvidia.com/object/cuda_home_new.html. [Online]

[26] Culler, D., Singh, J., Gupta, A.: Parallel Computer Architecture. A Hardware / Software Approach. The Morgan Kaufmann Series in CompuThe Morgan Kaufmann Series in Computer Architecture and Designter Architecture and Design, Handcover (1998)

[27] Curtiss, C.F., Hirschfelder, J.O.: Integration of Stiff Equations. Proceedings of the National Academy of Science of the Uniuted States of America **38**(3), 235–243 (1952)

[28] Davis, R.W., Moore, E.F.: A numerical study of vortex shedding from rectangles. Journal of Fluid Mechanics **116**, 475–506 (1982)

[29] Dubois, O., Gander, M.J.: Convergence Behavior of a Two-Level Optimized Schwarz Preconditioner. Lectures Notes in Computational Science and Engineering **70**, 177–184 (2009)

[30] Dubois, O., Gander, M.J., Loisel, S., ST-Cyr, A., Szyld, D.B.: The optimized Schwarz method with a coarse grid correction. SIAM Jurnal on Scientific Computing **34**(1), 421–458 (2012)

[31] Einfeldt, B., Munz, C., Roe, P., Sjogreen, B.: On godunov-type methods near low densities. Journal of Computational Physics **92**(2), 273–295 (1991)

[32] elsA: Home page. http://elsa.onera.fr/. [Online]

[33] Eltgroth, P., Bolstad, J., Duffy, P., Mirin, A., Wang, H., Wehner, M.: Coupled Ocean/Atmosphere Modeling on High-Performance Computing Systems. SIAM (1997)

[34] Emery, A.F.: An evaluation of several differencing methods for inviscid fluid flow problems. Journal of Computational Physics **2**(3), 306–331 (1968)

[35] Ernst, O.G., Gander, M.J.: Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods. Numerical Analysis of Multiscale Problems Lecture Notes in Computational Science and Engineering **83**, 325–363 (2012)

[36] Eymard, R., Gallouët, T., Herbin, R.: Finite Volume Methods. Handbook of numerical analysis (2006)

[37] Feireisl, E., Novotný, A., Petzeltová, H.: On the Existence of Globally De[U+FB01]ned Weak Solutions to the Navier-Stokes Equations. Journal of Mathematical Fluid Mechanics (2001)

[38] Gander, M.J.: Overlapping Schwarz Waveform Relaxation for Parabolic Problems. Proceedings of the 10th International Conference on Domain Decomposition, AMS, Contemporary Mathematics **218**, 425–431 (1998)

[39] Gander, M.J.: Schwarz Methods over the Course of Time. ETNA **31**, 228–255 (2008)

[40] Gander, M.J., Halpern, L.: Optimized Schwarz waveform relaxation methods for advection reaction diffusion problems. SIAM Journal on Numerical Analysis **45**(2), 666–697 (2007)

[41] Gander, M.J., Halpern, L.: Techniques for Locally Adaptive Timestepping Developed over the Last Two Decades. Domain Decomposition Methods in Science and Engineering XX, Lecture Notes in Computational Science and Engineering, Springer-Verlag (2012)

[42] Gander, M.J., Halpern, L., Kern, M.: A Schwarz Waveform Relaxation Method for Advection-Diffusion-Reaction Problems with Discontinuous coefficients and non-Matching Grids. Lectures Notes in Computational Science and Engineering **55**, 283–290 (2007)

[43] Gander, M.J., Halpern, L., Nataf, F.: Optimal Convergence for Overlapping and Non-Overlapping Schwarz Waveform Relaxation. Proceedings of the 11th International Conference on Domain Decomposition pp. 27–36 (1999)

[44] Gander, M.J., Halpern, L., Nataf, F.: Optimized Schwarz Methods. Domain Decomposition Methods in Sciences and Engineering pp. 15–29 (2001)

[45] Gander, M.J., Kwok, F.: Best Robin parameters for optimized Schwarz methods at cross points. SIAM Journal on Scientific Computing **34**(4), 1849–1879 (2012)

[46] Gander, M.J., Stuart, A.M.: Space-Time Continuous Analysis of Waveform Relaxation for the Heat equations. SIAM Journal on Scientific Computing **19**(6), 2014–2031 (1998)

[47] Gander, M.J., Wanner, G.: The Origins of the Alternating Schwarz Method. Lecture Notes in Computational Science and Engineering (2013)

[48] Gustafsson, B.: The Convergence Rate for Difference Approximations to Mixed initial Boundary Value Problems. Mathematics of Computation **29**(130), 396–406 (1975)

[49] Haeberlein, F.: Time-Space Domain Decomposition Methods for Reactive Transport Applied to CO_2 Geological Storage. PhD Thesis, University Paris 13 (2011)

[50] Haeberlein, F., Halpern, L.: Optimized Schwarz waveform relaxation for nonlinear systems of parabolic type. Proceedings of the 21 international conference on Domain Decomposition Methods (2012)

[51] Halpern, L.: Artificial Boundary Conditions for Incompletely Parabolic Perturbations of Hyperbolic Systems. SIAM **22**(5), 1256–1283 (1991)

[52] Halpern, L.: Absorbing Boundary Conditions and Optimized Schwarz Waveform Relaxation. Numerical Mathematics **43**(1), 001–018 (2003)

[53] Halpern, L.: Optimized Schwarz Waveform Relaxation: Roots, Blossoms and Fruits. Proceedings of the Eighteenth International Conference of Domain Decomposition Methods, Springer pp. 225–232 (2009)

[54] Halpern, L., Rauch, J.: Absorbing boundary conditions for diffusion equations. Numer. Math. **71**, 185–224 (1995)

[55] Halpern, L., Ryan, J., Borrel, M.: Domain decomposition vs. overset Chimera grid approaches for coupling CFD and CAA. ICCFD7 Proceedings (2012)

[56] Harten, A., D. Lax, P., Van Leer, B.: On Upwind Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws. SIAM **25**(1), 35–61 (1983)

[57] Hirsch, C.: Numerical Computation of Internal and External Flows. Volume 1. Fundamentals of Computational Fluid Dynamics. Elsevier (2006)

[58] Jeltsch, R., Pohl, B.: Waveform relaxation with overlapping splittings. SIAM Journal on Numerical Analysis **16**(1), 40–49 (1995)

[59] Keyes, D.E.: Domain decomposition in the mainstream of computational science. Proceedings of the 14 international conference on Domain Decomposition Methods (2002)

[60] Knoll, D.A., Keyes, D.E.: Jacobian-free Newton–Krylov methods: a survey of approaches and applications. Journal of Computational Physics **193**(2), 357–397 (2004)

[61] Lefebvre, M.: Algorithmes sur GPU pour la simulation numérique en mécanique des fluides. Phd thesis, Université de Paris XIII (2012)

[62] Lelarasmee, E., Ruehli, A.E., Sangiovanni-Vincentelli, A.L.: The waveform relaxation method for time-domain analysis of large scale integrated circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **1**(3), 131–145 (1982)

[63] Lions, J.L., Maday, Y., Turinici, G.: Résolution d'EDP par un schéma en temps pararéel. C. R. Acad. Sci. Paris Série I Math **332**, 661–668 (2001)

[64] Lions, P.L.: On the Schwarz alternating method I. In First International Symposium on Domain Decomposition methods for Partial Differential Equations. SIAM (1988)

[65] Lions, P.L.: On the Schwarz alternating method II: Stochastic Interpretation and Order Properties. SIAM (1989)

[66] Lions, P.L.: On the Schwarz alternating method III: A Variant for Nonoverlapping Subdomains. SIAM (1990)

[67] Liou, M.S.: A sequel to AUSM: AUSM$^+$. Journal of Computational Physics **129**(2), 364–382 (1996)

[68] Liou, M.S.: Ten years in the making - AUSM-family. AIAA-2001-2521 (2001)

[69] Liou, M.S.: A sequel to AUSM, part II: AUSM$^+$-up for all speeds. Journal of Computational Physics **214**(1), 137–170 (2006)

[70] Liou, M.S., Steffen, C.J.J.: A New Flux Splitting Scheme. Journal of Computational Physics **107**, 23–39 (1993)

[71] Magoulès, F., Roux, F.X.: Calcul scientifique parallèle. Dunod (2013)

[72] Marcinkowski, L., Cai, X.C.: Parallel Performance of Some Two-Level ASPIN Algorithms. Lectures Notes in Computational Science and Engineering **40**, 639–646 (2005)

[73] Martin, V.: Schwarz Waveform Relaxation Algorithms for the Linear Viscous Equatorial Shallow Water Equations. SIAM Journal on Scientific Computing **31**(5), 3595–3625 (2010)

[74] Moguen, Y., Delmas, S., Perrier, V., Bruel, P., Dick, E.: Intertia terms for all Mach number Godunov-type schemes: Behavior of unsteady solutions at low Mach number. Eighth Intenrational Conference on Computational Fluis Dynamics, to be published (2014)

[75] MPI: Home page. http://www.open-mpi.org/. [Online]

[76] Negrut, D.: High Performance Computing for Engineering Applications. GPU Course ME964, University of Winsconsin-Madison (2012)

[77] Nickolls John abd Buck, I., Garland, M.: Scalable Parallel Programming. Magazine queue - GPU Computing **6**(2), 40–53 (2008)

[78] Nordstrom, J., Svard, M.: Well-posed boundary conditions for the Navier–Stokes equations. SIAM Journal on numerical analysis **43**(3), 1231–1255 (2006)

[79] Novotný, A., Straškraba, I.: Introduction to the Mathematical Theory of Compressible Flow. Oxford Lecture series in Mathematics and Applications (2004)

[80] Okajima, A., K., K.: Numerical study on aeroelastic instabilities of cylinders with a circular and rectangular cross-section. Journal of Wind Engineering and Industrial Aerodynamics **46**, 541–550 (1993)

[81] O'Leary, D., White, R.: Multi-Splittins of Matrices and Parallel Solution of Linear Systems. SIAM J. Alg. Disc. Meth. **6**(4), 630–60 (1986)

[82] Ong, B., High, S., Kwok, F.: Pipeline Schwarz Waveform Relaxation. to be published in the Proceedings of the 22 international conference on Domain Decomposition Methods (2013)

[83] OpenMP: Home page. http://openmp.org/wp/. [Online]

[84] Ouvrard, H.: Simulation Numérique d'écoulements turbulents par les approches LES, VMS-LES et hybride RANS/VMS-LES en maillage non-structurés. Phd thesis, University Montpellier II (2010)

[85] Poinsot, T., Lele, S.: Boundary conditions for Direct Simulations of Compressible Viscous Flow. Journal of Computational Physics **101**(1), 104–129 (1992)

[86] Sanchez-Sanz, M., Belen, F., Velazquez, A.: Enery-Harvesting Microresonator Based on the Forces Generated by the Karman Street Around a Rectangular Prism. Journal of Microelectromechanical systems **18**(2), 449–457 (2009)

[87] Schwarz, H.: Uber einige Abbildungsaufgaben. Ges. Math. Abh. **11**, 65–83 (1869)

[88] Smith, B., Bjorstad, P., Gropp, W.: Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press (1996)

[89] Sod, G.A.: A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws. Journal of Computational Physics **27**(1), 1–31 (1978)

[90] Steger, J., Benek, J.: A 3-D Chimera Grid Embedding Technique. AIAA Paper **85** (1983)

[91] Steger, J., Dougherty, F., Benek, J.: A Chimera Grid Scheme. ASME Fluids Engineering Conference, Houston **5** (1983)

[92] Stegger, N.: A Numerical investigation of the Flow Around Rectangular Cylinders. Phd thesis, School of Mechanical and Materials Engineering, The University of Surrey, Guildford, United Kingdom (1998)

[93] Toro, E.F.: Riemann Solvers and Numerical Methods for Fluid Dynamics. Springer-Verlag Berlin Heidelberg (1999)

[94] Toro, E.F., Spruce, M., Spears, W.: Restoration of the Contact Surface in the HLL-Riemann Solver. Shock Waves **4**, 25–34 (1994)

[95] Van Leer, B.: Towards the Ultimate Conservative Difference Scheme; V. A Second-Order Sequel to Godunov's method. Communications in Computational Physics **32**, 101–136 (1979)

[96] Van Leer, B.: Flux-Vector Splitting for the Euler Equations. In Proceeding of the 8th International Conference on numerical Methods for Fluid Dynamics, Germany (1982)

[97] Van Leer, B.: Upwind and High-Resolution Methods for Compressible Flow: From Donor Cell to Residual-Distribution Schemes. Communications in Computational Physics **1**(2), 192–206 (2006)

[98] Woodward, P., Colella, P.: The numerical simulation of two-dimensional fluid flow with strong shocks. Journal of Computational Physics **54**(1), 115–173 (1984)

[99] Yee, H.C., Sandham, N.D., Djomehri, M.: Low-dissipative high-order shock-capturing methods using characteristic-based filters. Journal of Computational Physics **150**(1), 199–238 (1999)

[100] Yoon, S., Jameson, A.: An LU-SSOR Scheme for the Euler and Navier-Stokes Equations. AIAA Journal (1986)

Méthode de décomposition de domaine avec adaptation de maillage en espace-temps pour les équations d'Euler et de Navier–Stokes

Résumé

En mécanique des fluides, la simulation de phénomènes physiques de plus en plus complexes, en particulier instationnaires, nécessite des systèmes d'équations à nombre très élevé de degrés de liberté. Sous leurs formes originales, ces problèmes sont coûteux en temps CPU et ne permettent pas de faire une simulation sur une grande échelle de temps. Une formulation implicite, similaire à une méthode de Schwarz, avec une parallélisation simple par blocs et raccord explicite aux interfaces ne suffit plus à la résolution d'un tel système. Des méthodes de décomposition des domaines plus élaborées, adaptées aux nouvelles architectures, doivent être mises en place.

Cette étude a consisté à élaborer un code de mécanique des fluides, parallèle, capable d'optimiser la convergence des méthodes du type Schwarz tout en améliorant la stabilité numérique et en diminuant le temps de calcul de la simulation. Une première partie a été l'étude de schémas numériques pour des problèmes stationnaires et instationnaires de type Euler et Navier–Stokes. Deuxièmement, une méthode de décomposition de domaine adaptive en espace-temps, a été proposée afin de profiter de l'échelle de temps caractéristique de la simulation dans chaque sous-domaine. Une troisième étude a été concentrée sur les moyens existants qui permettent de mettre en oeuvre ce code en parallèle (MPI, OPENMP, GPU). Des résultats numériques montrent l'efficacité de la méthode.

Mots clés: mécanique des fluides numérique, équations d'Euler et de Navier–Stokes, méthode de décomposition de domaine, méthode adaptative de relaxation d'ondes, calcul haute performance, OpenMP, MPI, GPU

Adaptive Space-Time Domain Decomposition Methods for Euler and Navier–Stokes Equations

Abstract

Numerical simulations of more and more complex fluid dynamics phenomena, especially unsteady phenomena, require solving systems of equations with high degrees of freedom. Under their original form, these aerodynamic multi-scale problems are difficult to solve, costly in CPU time and do not allow simulations of large time scales. An implicit formulation, similar to the Schwarz method, with a simple block parallelisation and explicit coupling is no longer sufficient. More robust domain decomposition methods must be conceived so as to make use and adapt to the most of existent hardware.

The main aim of this study was to build a parallel in space and in time CFD Finite Volumes code for steady/unsteady problems modelled by Euler and Navier-Stokes equations based on Schwarz method that improves consistency, accelerates convergence and decreases computational cost. First, a study of discretisation and numerical schemes to solve steady and unsteady Euler and Navier–Stokes problems has been conducted. Secondly, an adaptive time-space domain decomposition method has been proposed, as it allows local time stepping in each sub-domain. Thirdly, we have focused our study on the implementation of different parallel computing strategies (OpenMP, MPI, GPU). Numerical results illustrate the efficiency of the method.

Keywords: Computing Fluid Dynamic, Euler and Navier–Stokes equations, Domain Decomposition Method, Adaptive Schwarz Waveform Relaxation method, high performance computing, OpenMP, MPI, GPU